UNIVERSITAT DE BARCELONA

Treball final de grau

GRAU D'ENGINYERIA INFORMÀTICA

Facultat de Matemàtiques i Informàtica

Universitat de Barcelona

NPC'S A FRACSLAND

Autor: Aleix Cots Molina

Directora: Dra. Anna Puig
Realitzat a: Departament de Matemàtiques i Informàtica

Barcelona, 27 de juny de 2018

Abstract

The best way to learn at school is the one which makes you forget that you are studying and allows you to enjoy what you learn. Based on this sentence, this project aims to provide a realistic basis for the videogame through the addition of artificial intelligence techniques, in order to facilitate the mathematical learning of students by making them enjoy a wider and complex gameplay when they interact with the various characters that make up the game.

Six different characters have been implemented in two different scenarios of the game, endowed with a set of AI techniques:

- *Steering*: A set of techniques that allow to perform the control of the motion of the characters to be implemented based on a single final vector of direction calculated from the influence of the weight of each particular technique on the total. In this way you get the direction that must follow the character who needs to take into account several aspects at the same time, such as the need to escape from a threat without leaving a specific area, for example.

- State machines: AI technique that allows to perform the control of all the states that form the behavior of the characters. It is a simple but effective technique, since it allows to model the different states belonging to a character with the transitions between these states.

- Behavior trees: A technique that allows to model complex and precise behaviors through an arboreal structure formed by two types of nodes, intermediate and leaf nodes. The first type is the internal structure of the tree, which allows to define the flow of it, such as the *Selector* and *Sequence* nodes, which allows to define a series of actions from which you can choose one at a time (*Selector*), or define a series of actions that will be executed one after the other until all success or one fails (*Sequence*). The second type is the leaf nodes, which allows to implement conditions and actions that the characters which possess a tree have to perform.

- Shared memory between behavior trees: AI technique that provides a tool to allow the behavior trees of all single-person characters to communicate between them. The goal is to synchronize trees in a more complex and precise way than the group *Steering* to perform complex group behaviors of humans that form formations, for example.

Resum

La millor forma d'aprendre a l'escola és la que et fa oblidar que estàs estudiant i et permet gaudir d'allò que aprens. Partint d'això, aquest projecte pretén proporcionar una base de realisme al videojoc mitjançant l'aplicació de tècniques d'intel · ligència

artificial, amb l'objectiu de facilitar l'aprenentatge matemàtic dels alumnes fentlos gaudir d'una jugabilitat més àmplia i complexa a l'hora d'interactuar amb els diversos personatges que composen el joc.

Han estat implementats sis personatges diferents en dos escenaris diferents del joc, dotats d'un seguit de tècniques d'IA:

- *Steering*: Conjunt de tècniques que permeten implementar el control del moviment dels personatges basat en un únic vector final de direcció calculat a partir de la influència del pes de cada tècnica particular sobre el total. D'aquesta forma s'obté la direcció en la què ha d'anar un personatge que necessiti tenir en compte diversos aspectes al mateix temps, com haver de fugir d'una amenaça sense sortir d'una àrea concreta, per exemple.

- Màquines d'estats: Tècnica d'IA que permet implementar un control de tots els estats que formen el comportament dels personatges. És una tècnica senzilla però efectiva, ja que permet modelar els diferents estats en els que es pot trobar un personatge amb les transicions entre aquests estats.

- Arbres de comportament: Tècnica d'IA que permet modelar comportaments complexes i precisos mitjançant una estructura arbòria formada per dos tipus de nodes, els intermitjos i els nodes fulla. El primer tipus són les estructures internes de l'arbre, que permeten definir el fluxe d'aquest, com per exemple els nodes *Selector* i *Sequence*, que permeten definir una sèrie d'accions de les quals se n'escollirà una cada cop (*Selector*), o definir una sèrie d'accions que s'executaran una darrera l'altra fins que s'executin totes o falli una (*Sequence*). El segon tipus són els nodes fulla, que permeten implementar condicions i accions que han de realitzar els personatges que tenen un arbre.

- Memòria compartida entre arbres de comportament: Tècnica d'IA que proveeix d'una eina per a permetre que els arbres de comportament de tots els personatges d'un sol tipus es puguin comunicar entre ells. L'objectiu és sincronitzar els arbres d'una forma més complexa i precisa que l'*Steering* grupal per a poder dur a terme comportaments grupals complexes propis d'humans que fan formacions, per exemple.

Agraïments

Vull agrair a l'Anna Puig, perquè més que una tutora ha estat una companya i mentora durant tot el desenvolupament d'aquest projecte. Sobretot vull agrair-li la seva dedicació, que hagi estat per mi sempre que ho he necessitat, ajudant, donant consells i proposant noves solucions i idees de forma constructiva i comprensiva quan no sabia com continuar o m'estressava i em bloquejava. Agraeixo moltíssim haver pogut treballar amb ella, ha estat realment un plaer.

Vull agrair a l'Aina Hernàndez, companya de projecte i amiga des de fa quatre anys, per haver estat allà amb mi, compartint els inicis del projecte, patint junts fins l'últim moment i sempre disposada a ajudar-me en tot el que li demanés sense demanar mai res a canvi.

Vull agrair a la Inmaculada Rodríguez, per haver estat pendent del meu projecte i per haver contribuït amb idees i propostes en moments determinats.

Per últim, vull agrair a la meva família i als meus amics per haver estat sempre al meu costat, suportant-me en els moments d'estrés i ajudant-me quan els he necessitat.

Índex

1 Introducció **1**

1.1 Motivació 1

1.2 Objectius generals 1

1.3 Objectius especifics 2

1.4 Planificació temporal 2

1.5 Organització de la memòria 2

2 Antecedents **4**

2.1 Fracsland 4

2.1.1 Què és Fracsland? 4

2.1.2 Personatges 4

2.1.3 Escenaris 6

2.1.4 Interfície 7

2.2 Tècniques d'IA per a controlar moviments 8

2.2.1 Moviments individuals: *Navigation Mesh* 9

2.2.2 Moviments individuals: Steering 10

2.2.3 Moviments en grup 11

2.3 Tècniques d'IA per a modelar comportaments 13

2.3.1 Comportaments individuals: FSM 13

2.3.2 Comportaments individuals: BT 13

2.3.3 Comportaments en grups 15

2.4 Conclusions 16

3 Anàlisi **18**

3.1 Anàlisi de l'actual versió de Fracsland 18

3.2 Nova proposta de *Mobs* 19

3.3 Escenaris a modificar 20

3.4 Requeriments tecnològics 21

4 Disseny de la IA dels NPCs **22**

4.1 Moviment amb Navigation Mesh 22

4.2 Moviment amb Steering 22

4.2.1 Moviments bàsics de *Mobs* 22

4.2.2 Moviments en grups: ocells, zebres 24

4.3 Comportament amb FSM 24
4.4 Comportament basat en Behavour Trees 24
4.4.1 Llops 25
4.5 Comportament basat an BTs compartits: Pirates 30
4.5.1 Pirates1: BTs sincronitzats 30
4.5.2 Pirates2: Formacions 31

5 Desenvolupament del joc **33**
5.1 Justificació de la tecnologia utilitzada 33
5.1.1 *Navigation Mesh* de Unity3D 33
5.1.2 UnitySteer 34
5.1.3 Projecte FSM 38
5.1.4 PandaBT 39
5.2 Modificació del joc inicial Fracsland 40
5.3 Diagrama de classes en Unity3D 41
5.4 Estratègies especials utilitzades en el desenvolupament del joc . . . 47

6 Resultats i Simulacions **50**
6.1 Ocells 50
6.2 Zebres 50
6.3 Lleons 53
6.4 Conills 54
6.5 Llops 55
6.6 Pirates 57
6.7 Videos de les simulacions 59

7 Conclusions i feina futura **60**
7.1 Conclusions 60
7.2 Línies futures 61
7.2.1 Intel · ligència artificial 61
7.2.2 Altres línies futures 62

1 Introducció

Aquest projecte està enmarcat dins del desenvolupament dels motors d'Intel · ligència Artificial integrat en els videojocs. Concretament, es focalitza en donar coportament als diferents personatges no jugables d'un videojoc (NPCs, *Non-Player Character, en anglès*), és a dir, el personatges que no estan controlats pel jugador.

Aquest projecte parteix d'un altre treball final de grau anterior dut a terme per Cristian Muriel, creador d'un videojoc seriós, o educatiu, orientat a l'aprenentatge de matemàtiques a l'educació primària [7]. La idea és extendre aquest primer joc donant-lo de més Intel · ligència Artificial per a fer-lo més jugable, explorant les diferents possibilitats de modelar el comportament intel · ligent dels diferents *NPCs*.

Aquest projecte es basa en els coneixements impartits dins del pla d'estudis del grau d'Enginyeria Informàtica de la UB, i especialment en les assignatures de GiVD, d'IA i IAD (aplicació de tècniques d'IA a videojocs).

1.1 Motivació

El projecte va sorgir de la necessitat de millorar el joc a nivell de realisme i de sensació d'inmersió en un món real al jugador. Normalment els jocs seriosos acaben focalitzant molt en la part educativa i poc en la part de joc. En el joc previ de Fracsland va ser un primer desenvolupament d'un joc divertit per aprendre fraccions, però amb un llarg cami per recórrer per a donar sensació d'inmersió en el joc. Una possibilitat d'extensió per a donar més realisme en el món on es desenvolupa el joc és afegir credibilitat en l'ecosistema que el forma, donant comportaments més realistes als animals que es troben en el joc, Aixií mateix, es poden incloure diferents estratègies de grup als humans enemics del jugador per a donar un component de més dificultat durant una partida.

Cal destacar que Fracsland és un videojoc per a que els alumnes l'utilitzin a l'escola, per tant ha de ser utilitzat en ordinadors amb alta probabilitat de tenir unes característiques tecnològiques no gaire altes, Així, l'abast del projecte queda limitat a utilitzar tècniques i algoritmes que permetin implementar credibilitat i realisme en els diferents *NPCs* i que, a més a més, no requereixin ordinadors potents.

1.2 Objectius generals

L'objectiu d'aquest projecte és l'anàlisi, la selecció, disseny i desenvolupament de diverses tècniques i estratègies d'intel · ligència artificial aplicades als personatges no controlats pel jugador dels videojocs, amb la finalitat de generar moviments i comportaments més realistes, propis d'humans capaços d'actuar conjuntament o d'animals que interactuen entre ells per moure's en grup o individualment.

1.3 Objectius especifics

Més concretament, els diferents objectius a realitzar són:

- Anàlisi, disseny i desenvolupament de tècniques per a modelar els comportaments individuals dels emphMobs (*NPCs* mòbils, generalment interactuen amb el jugador barallant-se), tal com menjar, beure, etc.
- Anàlisi, disseny i desenvolupament de tècniques per a modelar els comportaments grupals dels *Mobs*.
- Integració de les tècniques proposades en el joc Fracsland.
- Validació dels comportaments en l'entorn del joc.

1.4 Planificació temporal

Les primeres tasques que cal dur a terme són la desconnexió del servidor antereio del joc i la connexió al servidor local que es desenvolupi.

En segon lloc cal corregir tots els errors de la versió prèvia i proposar les noves millores. Per una banda les millores es basaran en les propostes de les tècniques de IA que s'hauran d'implementar als *Mobs*, i per altra banda seran les propostes de millores necessàries pel joc (com afegir nous *Mobs* per a aplicar tècniques d'IA) i de possibles millores que podrien millorar el joc, com la implementació del moviment de la càmera, etc.

Per últim caldrà analitzar, dissenyar i implementar totes les tècniques i estratègies proposades anteriorment.

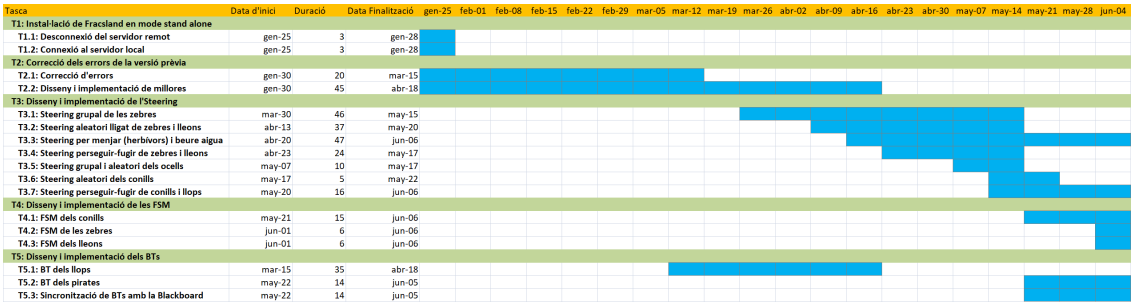


Figura 1: Diagrama de Gantt del projecte. Totes les setmanes des de l'última de Gener (25/01) fins la segona de Juny (06/06), ambdues incloses.

1.5 Organització de la memòria

La memòria està estructurada seguint un ordre cronològic:

- Capítol 2: Explicació de la versió inicial del joc de Fracsland i de totes les tècniques d'IA bàsiques disponibles per a millorar tant el moviment com el comportament dels *NPCs* dels videojocs.

- Capítol 3: Anàlisi del joc i propostes de millora de *NPCs* i escenaris, junt amb els requeriments tecnològics.
- Capítol 4: Explicació a nivell de disseny de totes les tècniques d'IA noves afegides al joc.
- Capítol 5: Justificació de les tecnologies utilitzades que implementen les tècniques explicades en el capítol anterior junt amb l'explicació a nivell d'implementació de totes les modificacions fetes al joc.
- Capítol 6: Exposició dels resultats obtinguts amb les simulacions fetes per provar totes les tècniques d'IA implementades a cada *NPC*.
- Capítol 7: Explicació dels objectius assolits, conclusions sobre els avantatges i els inconvenients de les tècniques d'IA utilitzades i possibles continuacions per altres projectes, tant d'IA com d'altres àmbits.

2 Antecedents

Prèviament al desenvolupament del projecte, cal analitzar tant la versió inicial de Fracsland com les diferents tècniques d'IA per a modelar moviments i comportaments de les que es pot disposar.

2.1 Fracsland

2.1.1 Què és Fracsland?

Fracsland [7] és un joc de tipus RPG (Role-Playing Game) en què el jugador és un nàufrag que arriba a una illa on pot realitzar missions encomenades pel líder de l'illa, en Lord Barus, i aconseguir peces per a reconstruir el seu vaixell per a poder sortir d'allà.

Està enfocat a ser un joc educatiu, per tant les missions són dutes a terme utilitzant diferents fraccions matemàtiques que el jugador ha d'utilitzar correctament per tal d'assolir els objectius imposats pel propi joc. Fracsland també compte amb una connexió a un servidor del qual depèn per obtenir les missions i les fraccions necessàries. Aquest servidor és controlat pel professor, encarregat de posar quines fraccions són les que han d'aprendre els alumnes. A banda d'això el servidor també s'encarrega de la persistència de dades guardant les partides dels alumnes que hi juguen ordenat per classes i cursos.

Aquest joc va ser desevolupat en un treball de grau d'enginyeria informàtica per Cristian Muriel [7] i va tenir una bona acceptació entre nens i nenes de sisè curs de primària.

2.1.2 Personatges

Com qualsevol joc RPG aquest joc també té NPCs (*Non-Personal Character* en anglès), és a dir, personatges controlats pel propi joc que poden interactuar amb el jugador, els quals es divideixen en dos grups.

En primer lloc estan els NPCs de l'illa principal:

- Lord Barus: És l'encarregat de controlar que tot vagi bé a l'illa i d'interactuar amb l'usuari per a donar-li missions (Figura 2a).
- Venedor: Només interactua amb l'usuari venent-li armes (Figura 2b).
- Altre: No interactuen amb l'usuari, únicament estan a l'illa fent les seves feines (Figura 2c).
- John The Builder: NPC final del joc (Figura 2d).
- Animals de granja: NPCs de la granja que ha de cuidar el jugador (Figures 2e 2f 2g).

(a) Lord Barus

(b) Venedor

(c) Altre NPC

(d) John The Builder

(e) Vaca de la granja

(f) Gallina de la granja

(g) Conill de la granja

Figura 2: NPCs de Fracsland

En segon lloc estan els NPCs que es mouen lliurement per tot el mapa i interactuen amb l'usuari de forma violenta quan aquest s'apropa, atacant-lo o fugint, és a dir, d'una forma més dinàmica. Aquests últims són els que l'usuari ha de matar per tal d'assolir els objectius de les missions o simplement per evitar ser assassinat, i són els següents:

- Llops i pirates: Es mouen d'un punt a un altre aleatòriament per tota una zona concreta i ataquen el jugador quan aquest està prou aprop (Figures 3a i 3b).
- Lleons: També ataquen al jugador quan aquest està prou aprop però romanen quiets, sense moure's per la zona en la que es troben (Figura 3c).
- Ocells: Només es mouen de forma completament aleatòria d'un punt a un altre del cel (Figura 3d).

Tot i que ambdós tipus de personatges són NPCs, per poder diferenciar-los s'anomena NPCs als del primer tipus, és a dir, que el Lord Barus i el venedor són NPCs; mentre que l'altre tipus col · loquialment és anomenat *Mob*, de l'anglès *Mobile*, fent referència a que usualment són els NPCs mòbils del joc.

(a) Llop

(b) Pirates

(c) Lleo

(d) Ocell

Figura 3: *Mobs* de Fracsland

2.1.3 Escenaris

El joc està constituït per quatre escenaris (veure Figura 4):

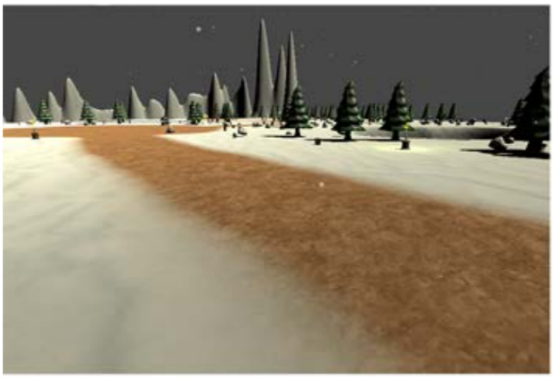
- **Fracstown**: Escenari principal de l'illa format per una plaça principal (designada com el punt de partida del jugador) on hi ha unes cases i un NPC que ven armes al jugador. També està format per quatre camins, tres dels quals condueixen als demés escenaris. L'últim porta directament al Lord Barus. A banda també es poden veure altres NPCs que treballen amb fusta, amb els quals no es pot interactuar (Figura 4a).

- **Beach**: Escenari que simula l'entorn d'una platja. Està dividit en dues parts per un tros de mar que el travessa ja que un dels objectius del joc és construir un pont per tal d'accedir a l'altra banda de l'illa. La part amb la que el jugador pot interactuar està formada únicament per un petit tros de l'illa que conté els lleons, les palmeres que utilitzarà el jugador per a obtenir la fusta requerida en les missions i el vaixell en el que va arribar a l'illa (Figura 4c).

- **Forest**: Escenari que simula l'entorn d'un bosc nevat. Està format per un camí central que condueix directament cap al centre de l'escenari. El centre és una petita plaça que es divideix en dos camins, un cap a l'esquerra, sense destí, i un altre cap a la dreta que porta directament al NPC final del joc, el pirata *John the Builder*. A part d'això l'escenari està constituït per elements de estàtics i de vegetació com arbres, roques, una casa, un pou i un llac, i per altra banda també conté elements mòbils: llops i pirates (Figura 4b).

- **Farm**: Escenari que simula un entorn de granja, constituït per tres animals (gallina (2f), conill (2g) i vaca (2e)) i un hort que proveeixen de menjar a tots els habitants de l'illa. També es centren algunes missions del joc (Figura 4d).

(a) FracTown



(b) Forest

(c) Beach

(d) Farm

Figura 4: Escenaris de Fracsland

2.1.4 Interfície

La interfície del joc és molt similar a la de qualsevol joc RPG. Inicialment trobem la que permet a un jugador introduir el seu nom i contrasenya (veure Figura 5).

A continuació, un cop feta la connexió amb el servidor, es mostra una taula amb les classes a les que l'usuari està registrat i que ha configurat el professor prèviament (veure Figura 6).

Per últim, la interfície de comunicació amb el joc és prou senzilla ja que consta únicament dels elements més bàsics dels jocs RPG. En primer lloc hi ha la informació bàsica i general del jugador a la part superior esquerra: nom, cara del personatge, vida i nivell. En segon lloc hi ha una taula a la part dreta que conté el mapa amb una fletxa que indica la posició del jugador, i altre informació del jugador com els diners i diamants que té, i les missions que té actives en aquell moment (o indicacions per anar a on pot adquirir missions si no en té en aquell moment.

Després també conté una barra horitzontal a la part inferior de la pantalla que mostra els sis primers elements de l'inventari junt amb els *shortcuts* necessaris per beure pocions curatives (Q), obrir un panell que mostra l'inventari complet (I) i obrir un panell que mostra els estats del jugador com la força, la velocitat i la vida (K). Per últim, a la part inferior esquerra, es pot trobar un botó que permet sortir del joc (veure Figura 7).

Figura 5: Interfície d'inici

Figura 6: Interfície de selecció de classe

2.2 Tècniques d'IA per a controlar moviments

Pel control del moviment dels *Mobs*, les tècniques a la nostra disposició són les següents:

- *Navigation Mesh*: Defineix l'àrea (o àrees) per la qual s'han de moure els *Mobs*.

Figura 7: Interfície de joc

- *Steering*: Controla la gran majoria d'aspectes del moviment que pot tenir un *Mob* (moviment individual) o diversos *Mobs* del mateix tipus (moviment grupal).

2.2.1 Moviments individuals: *Navigation Mesh*

La *Navigation Mesh* és una tècnica que permet definir l'àrea dintre del mapa, per la que tots els personatges es poden moure amb seguretat en tant que tots els elements mòbils puguin anar a zones perilloses o inaccessibles. També és utilitzada per a poder determinar quins objectes estàtics dintre del mapa queden fora de la *Navigation Mesh*, el que permet fer, no només que els elements mòbils tinguin en compte aquests objectes estàtics i no els travessin, sinó que també ajuda a preveure tots els obstacles que pot haver-hi en un recorregut concret que hagi de fer el *Mob* des d'un punt a un altre, com pot ser una roca, un arbre o una casa.

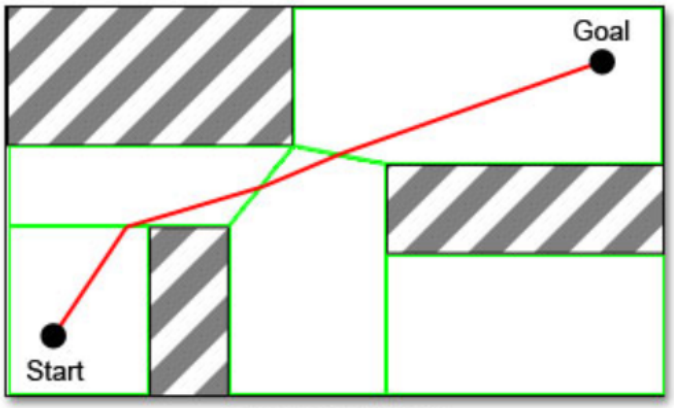


Figura 8: Exemple de *Navigation Mesh*

2.2.2 Moviments individuals: Steering

El moviment (o *Steering* en anglès) consisteix en un conjunt d'estratègies de combinar diferents tècniques de càlcul de moviments que permeten simular uns comportaments o uns altres utilitzant un sistema diferent a tots els altres mètodes de modelatge. Es basen en calcular un vector final, a partir dels vectors de moviments calculats per diferents tècniques aplicades al *Mob*. Cada vector calculat es ponderà amb un pes segons la seva importància en el moviment. L'avantatge principal d'aquest sistema radica en què no cal definir una llista de prioritats i decidir en cada moment quin moviment fa el *Mob*, sinó que tots els diferents comportaments calculen el vector que més els interessa a cadascun i depenent de la importància que tingui cadascun en aquell precís moment s'agreguen tots, ponderats generant un vector final, que és el que dictarà la direcció del *Mob*.

Com a exemple es defineix un *Mob* que té definides dues estratègies, la de fugir i la de zona (*Mob* lligat a una zona per evitar que s'allunyi). Si en un cert moment el *Mob* necessita fugir, perquè hi ha alguna amenaça suficientment aprop, es pujarà el pes del vector de fugir i així el vector final estarà fortament lligat a la fugida i el *Mob* es mourà fugint de l'amenaça. Com que també té associada la tècnica que el lliga a una zona determinada (àrea de la qual no hauria de sortir) doncs fugirà fins que s'allunyi massa d'aquesta zona, aleshores el pes s'anirà incrementant fins que sigui prou alt com per a modificar el vector final i aquest determini que la direcció a fugir és cap a la zona en la que el *Mob* ha de romandre. A mesura que s'apropi al centre de la zona el pes d'aquesta última tècnica anirà disminuint, permetent a la tècnica de fugir controlar gaire bé el 100% de la direcció final. Quan passi el perill perdrà tot el pes i aleshores les demés tècniques tornaran a influir en el vector final com ho feien prèviament a l'atac.

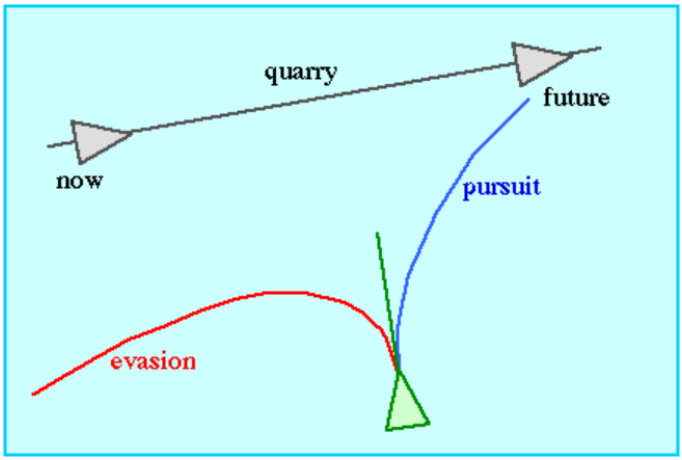


Figura 9: Exemple de *Steering* (Fugir en vermell, perseguir en blau)

Les estratègies de moviment més utilitzades a nivell individual són les següents:

- ***Wander***: Aquesta és l'estratègia més bàsica de l'*Steering*, aplicada a qualsevol *Mob* que hagi de tenir un comportament de moviment aleatori, sense rumb

fixe ni cap tipus de restricció. Únicament s'utilitza la tècnica de *Wander* per tal de calcular un vector aleatori.

- ***Wander* restringit**: Aquesta estratègia és utilitzada per a controlar el moviment de *Wander* en una certa àrea tancada, és a dir, utilitzant les tècniques de *Wander* pel vector aleatori, i *Tether* per a calcular un vector que apunti cap al centre de l'àrea dessignada amb una força proporcional al seu pes.

- **Assolir un punt**: Estratègia relativament senzilla basada en una sola tècnica que calcula el vector de direcció necessari per tal de poder arribar a aquest punt.

- **Perseguir**: Aquesta estratègia ja és una mica més complexa degut a que el vector calculat a la tècnica de perseguir ha de tenir en compte que l'objectiu es mourà, és a dir, que ha de preveure la direcció en la que es mou i calcular el vector resultant, tal i com es pot observar en la línia blava dibuixada a la figura 9. També ha d'implicar la distància a la què es troba de l'ojbectiu en el càlcul del vector per a ralentitzar la seva velocitat quan s'apropi. En molts casos també s'aplica l'estratègia de *Tether* per a evitar que el *Mob* s'allunyi massa de la zona en la que ha d'estar, cosa que influeix en el vector total de l'estratègia.

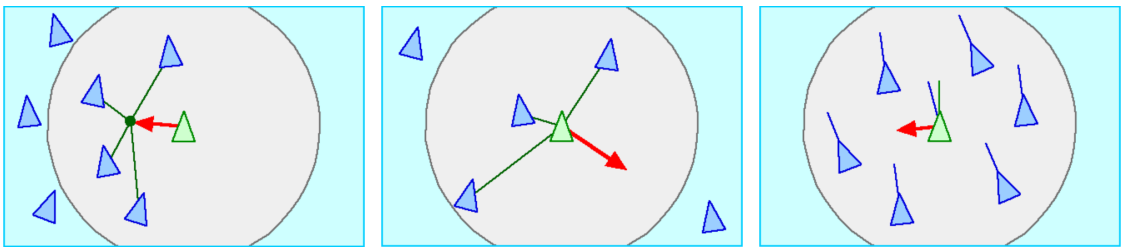
- **Fugir**: Semblant a l'estratègia anterior en gaire bé tots els aspectes, ja que també calcula el vector tenint en compte la posició de l'amenaça (figura 9, la distància a la que es troba (quant l'amenaça sobrepassi una distància mínima ha de fugir) i la direcció de l'amenaça si es mou, és a dir, la predicció de la seva direcció amb un temps determinat. L'única diferència radica en què el càlcul del vector es fa just en la direcció contrària a la que es troba l'amenaça, per tal d'allunyar-se d'ella. De la mateixa forma que l'estratègia anterior, aquesta també inclou en la majoria dels casos el *Tether* per a evitar que el *Mob* s'allunyi massa de la zona en la que ha de romandre.

2.2.3 Moviments en grup

En aquesta secció s'estudia en el comportament que poden tenir els diferents *Mobs* en grup, és a dir, tenint en compte els altres *Mobs* del mateix tipus que hi ha al seu voltant. Per tal de modelar aquests comportaments grupals hi ha diferents tècniques [11]. Entre elles cal distingir el *flocking*, usualment relacionat amb els moviments en manada de diferents animals com els ocells o les zebres i les formacions, típiques en grups considerats més intel · ligents com els humans, capaços d'agrupar-se per organitzar atacs, per exemple.

Les estratègies d'*Steering* utilitzades en el moviment grupal són diferents a les anteriors ja que totes es basen en una estratègia bàsica de grup basada en tres tècniques (Figura 10).
Són tècniques que s'apliquen individualment a cada *Mob*, però que, aplicades a tots els *Mobs* del grup, acaben donant un moviment grupal:

- **Cohesió**: Tècnica encarregada de trobar els *Mobs* més propers del mateix tipus i calcular el vector necessari per a fer que el *Mob* s'apropi més al grup, la zona on més *Mobs* del mateix tipus hi ha, per tal de cohesionar-lo més (Figura 10a).
- **Separació**: Tècnica oposada a l'anterior, utilitzada amb l'objectiu de regular la distància que hi ha entre els *Mobs* del grup per tal d'evitar que s'ajuntin massa. Comprova la zona on hi ha menys *Mobs* del mateix tipus i calcula el vector des de la posició del *Mob* cap a aquella direcció, d'aquesta forma finalment s'aconsegueix un efecte de separació amb tots els *Mobs* (Figura 10b).
- **Alineació**: Tècnica emprada per a aconseguir que els *Mobs* es moguin en una direcció conjunta. També calcula un vector, però aquest només està influenciat per la direcció que tenen tots els altres *Mobs* (Figura 10c).



(a) Cohesió (b) Separació (c) Alineació

Figura 10: Tècniques de l'estratègia bàsica de moviment grupal

Un cop creada aquesta estratègia es pot combinar amb estratègies individuals per tal de formar-ne de més complexes com les següents:

- ***Wander* grupal**: Estratègia utilitzada per simular el moviment aleatori de *Wander* amb molts *Mobs* movent-se al mateix temps, com un estol d'ocells que es mou en la mateixa direcció, tot i ser erràtica i canviant, sempre a una distància aproximada els uns dels altres.
- ***Wander* grupal restringit**: De la mateixa forma que l'exemple individual, l'estratègia del *Wander* grupal també pot es pot utilitzar amb el *Tether*, per a fer que el grup de *Mobs* que es mou conjuntament no s'allunyi massa de l'àrea dessignada.
- **Assolir un punt**: Aquesta no és una estratègia de grup pura pel fet que tots no es poden dirigir cap a un punt, sinó seria una estratègia individual. L'única estratègia grupal possible seria amb un líder de grup que hagi d'assolir el punt i que tots els demés el segueixin utilitzant l'estratègia grupal, que simularia el moviment en manada.
- **Perseguir**: Estratègia anàloga a la individual, amb la diferència que aquesta té en compte als diferents elements del grup per tal de perseguir un objectiu.

El càlcul del vector resultant és la suma del càlcul dels vectors d'ambdues estratègies (la grupal base i la de perseguir).

- **Fugir**: De nou, anàlogament a l'estratègia homòloga de l'apartat anterior, aquesta estratègia calcula el vector conjunt de l'estratègia grupal base i la de fugir, cosa que fa un efecte de fugida en massa.

2.3 Tècniques d'IA per a modelar comportaments

Fins aquest punt s'han introduït les diferents tècniques emprades en el control del moviment dels *Mobs*, però a vegades cal controlar altres aspectes de la IA com són els comportaments dels personatges. Aquests comportaments es poden modelar com estats o com comportaments específics, i poden ser individuals o col · lectius. Existeixen moltes tècniques de IA per modelar comportaments [10]. Concretament en aquest projecte s'analitzen les FSM i els BTs.

Anàlogament als anteriors apartats on una part del control del moviment era individual i l'altra s'encarregava de modelar el comportament grupal, les tècniques de modelatge del comportament també es poden dividir en individuals i grupals: a nivell individual tenim les FSM i els BT, explicats en els següents apartats.

2.3.1 Comportaments individuals: FSM

FSM (Finite State Machine, en anglès), o màquina d'estats finits [10], és la tècnica més utilitzada en el modelatge del comportament ja que és una eina senzilla i molt intuïtiva, útil per a definir un conjunt d'estats i les transicions entre ells.

Una FSM és un graf on els nodes representen "estats", concretament en el cas del joc, poden modelar estats del *Mob*. Cada estat representa un comportament específic (o configuració interna) i només existeix un únic estat actiu en un moment determinat. Aquests nodes estan connectats per arestes o "transicions", enllaços directes entre estats encarregats de canviar d'un estat a un altre (canviar l'estat actiu) quan es compleixen unes certes condicions.

Un exemple d'una FSM seria el cas del vigilant (veure Figura 11) amb els estats ***Patrol***, ***Investigate***, ***Attack*** i ***Flee***. El vigilant comença patrullant (*Start*→*Patrol*) i vigila els possibles enemics. Quan el vigilant escolta algun soroll passa automàticament a investigar-ho (*Patrol*→*Investigate*). Si no troba res durant la investigació, torna a patrullar (*Investigate*→*Patrol*). En cas que vegi l'enemic durant la investigació (*Investigate*→*Attack*) o durant el patrullatge (*Patrol*→*Attack*) passa directament a atacar-lo. Finalment, si durant l'atac queda greument ferit, el vigilant fugirà per salvar la vida (*Attack*→*Flee*).

2.3.2 Comportaments individuals: BT

BT (*Behavior Tree*, en angles), o arbre de comportament, és una tècnica de IA que, a diferència de la FSM descrita anteriorment, no defineix una sèrie d'estats

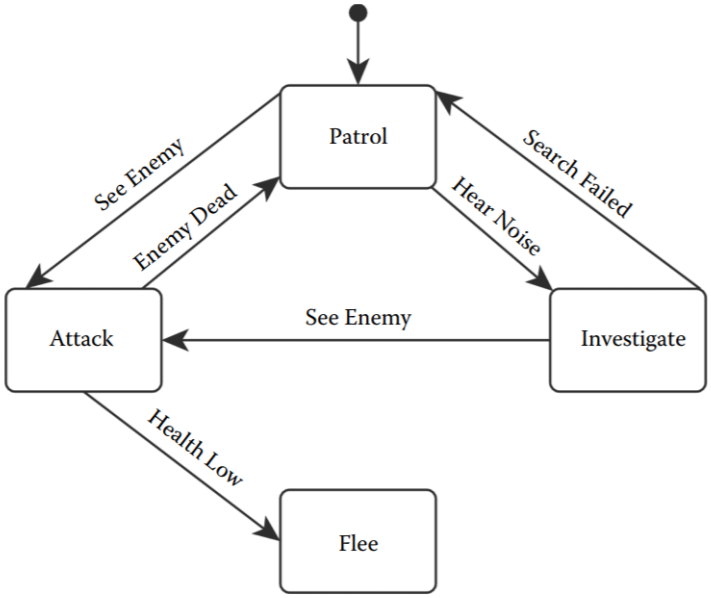


Figura 11: FSM d'un vigilant amb els estats de Patrol, Investigate, Flee i Attack. El punt de la part superior indica que la FSM comença per aquell estat.

amb transicions, sinó un conjunt d'accions que realitza el *Mob* quan es compleixen una sèrie de condicions organitzades en un arbre [8]. Modelar el comportament consisteix bàsicament en travessar l'arbre i comprovar a cada node si és possible realitzar l'acció que codifica. Si s'acompleix s'executa una acció, en cas contrari es passa al següent node de l'arbre, fins que troba la que sí s'acompleix.

Hi ha dos tipus de node fulla en els BT:

- Els que evaluen condicions (donen *Success* o *Failure*).
- Els que executen accions derivades de l'avaluació de condicions. Poden ser canvis en els atributs del *Mob*, en el seu entorn, moviment, animacions, etc. També poden donar *Success* o *Failure*.

A banda d'aquests nodes fulla, es defineixen els nodes intermitjos de l'arbre que controlen la lògica entre diferents nodes fulla:

- **Seqüència**: Node intermig que permet fer agrupacions dels seus nodes fill mitjançant la concatenació d'una sèrie d'accions, definides en els seus fills, les quals s'executaran una darrere l'altra fins que totes retornin l'estatus *Successful* o fins que una d'elles falli i retorni un *Failure*. Això, proporciona una eina que pot ser útil a l'hora de formar plans seqüencials a seguir ja que executa tots els fills fins que un retorna *Failure*.
- **Selector**: Node intermig utilitzat normlament de node arrel de l'arbre ja que el seu funcionament es basa en recórrer els diferents nodes fills (comportaments) que hi ha al BT, executant-los i obtenint-ne un status, però a

diferència de la seqüència, el selector recorre tots els seus nodes fills mentre retornin un *Failure* fins que troba un node que retorna un *Successful*. És a dir, va a cercar el primer fill que retorna *Successful*.

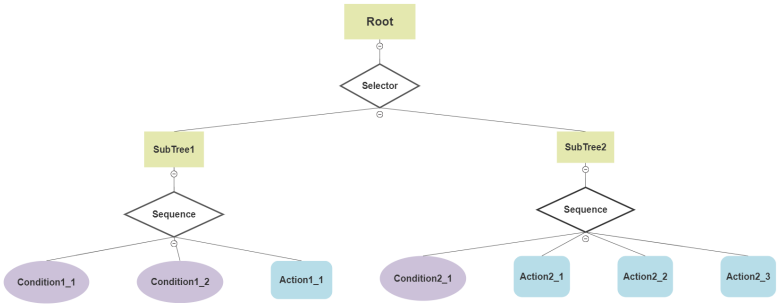


Figura 12: Exemple de BT

A banda d'aquests dos principals tipus de node també existeixen altres tipus de nodes intermitjos que permeten optimitzar el fluxe de l'arbre:

- **Paral · lel**: Node similar a la seqüència, però aquest executa tots els nodes fills al mateix temps fins que tots retornen *Success* o un d'ells retorna *Failure*.
- **Aleatori**: Node que selecciona aleatòriament un dels seus nodes fills i l'executa. També permet dessignar un pes per a cada node fill, d'aquesta forma quan més alt sigui el pes d'un fill més probabilitats hi haurà de que sigui escollit.
- **While i RepeatN**: Nodes que executen un node fill fins que aquest falla (While) o un nombre determinat de cops (RepeatN).
- **Repeat N**: Node que executa un node fill N cops.

En la figura 12 es veu un exemple de BT que modela [falta fer un BT]

2.3.3 Comportaments en grups

Tot i que els FSM es poden usar per sincronitzar diferents *Mobs*, els BTs donen més flexibilitat per modelar comportaments grupals o de manades degut a que permeten assolir un grau de disseny i implementació de la IA més flexible i complex.
Així doncs, com a tècnica de comportament grupal s'ha considerat els BT col · laboratius [9].

A diferència dels BT simples, que només tenen en compte la informació interna del *Mob* i els diferents "estímuls"que poden rebre de l'entorn, els BT col · laboratius comparteixen una memòria o *blackboard* a la qual poden accedir per tal de dur a terme accions grupals, sincronitzades, amb l'objectiu de crear estratègies grupals com formacions de diferents *Mobs* en determinades posicions, entre d'altres (veure figura 13).

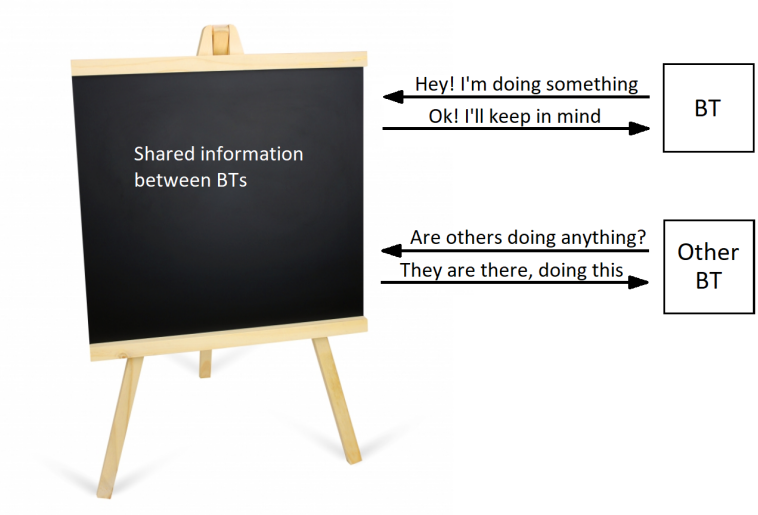


Figura 13: Exemple de Blackboard compartida entre BTs

2.4 Conclusions

Arribats a aquest punt, havent presentat diferents tècniques de modelatge del comportament i del moviment, es pot concluir en relació al moviment, l'ús de la *Navigation Mesh* unit a diferents estratègies de *Steering* permet modelar de forma realista el moviment assegurant que els *Mobs* es mouran en zones segures i segons les seves prioritats.

En relació al comportament, el fet de poder escollir entre dos tipus de tècniques per modelar-lo, condueix a reflexionar sobre quins casos es poden aplicar les tècniques de FSM i BT.

Tal i com s'ha dit prèviament, les FSM són tècniques senzilles i útils per tal de modelar diferents estats d'un *Mob* els quals queden connectats entre ells mitjançant les transicions, mentre que els BT faciliten el modelatge del comportament a nivell de condició-acció, sense haver de passar abans per altres estats utilitzant transicions, cosa que permet dissenyar tècniques més flexibles i extensibles.

Traslladant aquesta informació a l'exemple del vigilant, es pot deduir que la FSM seria útil per a modelar els diferents estats pels quals es "mou"el vigilant (patrullar, vigilar, atacar i fugir), tal i com es pot veure a la Figura 11, ja que no és un comportament que requereixi ser molt específic, sinó més aviat s'enfoca en controlar els diferents estats globals que pugui haver-hi en el comportament del vigilant. Per altra banda, i seguint amb el mateix exemple, si cal afegir estats al vigilant s'hauria de modificar el FSM per tal d'incloure'l amb totes les transicions necessàries des dels altres estats. Quan el comportament es fa més complex, els BTs semblen més adients. Si sorgeix la necessitat de fer que el comportament del vigilant sigui més complex en algún dels estats, seria viable afegir un BT que s'encarregués de definir una sèrie d'accions amb condicions, com per exemple un BT que modelés l'atac per fer la lluita més intel · ligent (atacar, esquivar, esperar, comprovar ferides, fugir, perseguir, etc.).

Per últim es pot concluir que els BT compartits és una bona tècnica per modelar el comportament grupal ja que, com s'explica a l'apartat anterior, els BT permeten dissenyar i implementar una IA més complexa que les altres tècniques, com un atac grupal sincronitzat. Tot i així cal remarcar que les FSM també poden arribar a sincronitzar dos *Mobs* diferents encara que no sigui aquest el seu objectiu, com per exemple un estat que sigui "fugir", el qual sincronitzarà la presa amb el predador simulant així un comportament.

3 Anàlisi

Un cop vista la versió inicial del videojoc de Fracsland i exposat tot el material disponible per tal d'afegir intel · ligència artificial al videojoc de Fracsland, el següent pas consisteix en l'anàlisi del videojoc per trobar on poden ser implementades totes aquestes tècniques d'intel · ligència artificial i quins requeriments comporten.

3.1 Anàlisi de l'actual versió de Fracsland

El joc de Fracsland desenvolupat en el projecte [7] està format per quatre escenes principals per les quals es pot moure l'usuari: l'illa principal, el bosc, la platja i l'hort. Aquest joc està desenvolupat en *Unity3D* i està controlat per un GameController encarregat de controlar tots els aspectes generals del joc, com per exemple carregar l'escena corresponent en passar per un portal determinat o iniciar la partida. Un altre aspecte important a tenir en compte és el fet que el joc requereix d'usuari i contrassenya per a poder jugar-hi. El joc està connectat a un servidor que conté la informació de tots els jugadors que hi estan enregistrats, és a dir, com tots els jocs MMORPG d'avui dia, amb la diferència que aquest no és multijugador. Aquest servidor també dóna la persistència del joc.

Fracsland posseeix una IA prou bàsica, que únicament permet a alguns dels personatges com els pirates i els llops al bosc moure's d'un lloc a un altre aleatori sense aturar-se, i perseguir el jugador quan aquest està prou aprop. Un altre *Mob* que podem trobar al joc és el lleó, a l'illa, només té la part que els permet moure's quan el jugador està aprop, ja que en qualsevol altre cas roman quiet. Els altres dos escenaris estan formats pels NPCs humans que ajuden el jugador (escena principal) o per tres *Mobs* engabiats (hort).

Les possibles millores observades són:

- **Càmera**: En la versió inicial de Fracsland, la càmera està modelada en tercera persona i segueix al jugador a una certa distància. Es por modificar aquesta càmera per a què deixi de ser completament estàtica, ja que ara està sempre a la mateixa posició relativa al jugador, i permeti el moviment rotatori en esfera al voltant del jugador (utilitzant el ratolí), fet que milloraria molt l'experiència de joc.

- **Navigation Mesh**: Afegir una *Navigation Mesh* seria doblement útil ja que facilitaria molt el control dels *Mobs* en relació als límits del terreny als quals poden arribar i, a més a més, permetria afegir més realisme ja que es tindrien en compte els objectes del terreny, fet que evitaria que tots els personatges que es mouen (tant NPCs, com jugador) travessessin els objectes com si fossin fantasmes.

- **Afegir *Mobs***: Afegir més *Mobs* a la platja o al bosc li proporcionaria al joc més realisme ja que els llops i els lleons són depredadors, però fins ara viuen completament sols en els seus respectius hàbitats.

- **Terreny**: Afegir més *Mobs* implica modificar el terreny necessàriament en dos aspectes. El primer de tots és ampliar-lo per a tenir més espai pels nous *Mobs*, i el segon consisteix en afegir elements com aigua dolça per a què puguin beure els animals o herba per a que puguin menjar els herbívors.

Una millora general que es pot aplicar a tot el joc consistiria en ampliar la IA dels *Mobs* utilitzant tècniques més específiques com les esmentades en el capítol anterior per tal de fer que el joc sigui més realista, més jugable. Una altra millora seria implementar el multijugador per tal de que els alumnes poguessin jugar junts per tal de dur a terme missions conjuntament, treballar en equip, etc.

Un punt important a considerar en compte és que per a poder testejar el joc en incorporar-li aquests canvis i nous personatges, com no afecta a la part educativa del joc, es pot es pot desconnectar del servidor i deixarel joc en local.

3.2 Nova proposta de *Mobs*

Per a decidir els *Mobs* a modelar, cal tenir en compte els diferents hàbitats dels que consta el joc i establir cadenes alimentàries a cadascun d'ells. A continuació es detallen els *Mobs* segons les diferents regions.

A la **platja** tenim els següents animals:

- **Ocells**: Volen en bandada, sense topar-se entre ells i sense objectius concrets.
- **Zebres**: Les zebres són animals herbívors que sempre van en bandada. Utilitzen una gran part del temps en menjar tot tipus de plantes que troben arran de terra, beure aigua i tenir en compte els lleons que les volen caçar. Els mascles acostumen a ser territorials, per tant sempre es troben dins del seu territori.
- **Lleons**: Els lleons són animals carnívors que, tot i que viuen en grup, acostumen a ser grups reduïts o fins i tot solitaris (mascles) en alguns casos. També són territorials, han de beure aigua i alimentar-se, però a diferència dels herbívors que dediquen tant de temps a menjar, quan els lleons han caçat una presa (acostumen a ser més grans que ells) i s'alimenten, poden estar setmanes sense tornar a menjar.

En el **bosc**:

- **Conills**: Els conills són animals generalment herbívors que s'alimenten de pràcticament tot el que troben pel terra, majoritàriament plantes. Acostumen a viure en caus i sortir quan han de menjar o beure aigua.
- **Llops**: Els llops són animals carnívors que poden arribar a estar unes dues setmanes sense menjar, tot i aque acostumen a beure molta aigua per evitar problemes urèmics (residus tòxics a la sang que no han estat filtrats degudament pels ronyons). Són territorials, per tant difícilment surten del seu territori, només per menjar quan és realment necessari.

- **Pirates**: Els pirates no tenen un comportament genèric en aquest cas, ja que el fet de ser humans genera una quantitat d'opcions gaire bé infinita. Tot i així normalment són agressius amb els altres humans que se'ls puguin apropar, fins a considerar-lo enemic i aleshores actuar en grup rodejant-lo fent usant diferents formacions d'atac.

3.3 Escenaris a modificar

Per incloure tots els *Mobs* cal fer les modificacions a tots els escenaris per tal d'evitar que els *Mobs* o el jugador surtin del terreny o vagin per zones perilloses o no realistes com les zones d'aigua o travessar objectes sòlids com cases.

Per tant cal modificar les zones definides a Fracsland (veure secció 2.1) per a facilitar la inclusió dels nous *Mobs* amb les noves tècniques de IA que requereixen més terreny i més elements (aigua i menjar pels herbívors):

- **Platja**: Cal ampliar la platja perquè ara és prou petita i només conté lleons. La idea principal consisteix en ampliar el terreny per a generar més espai pels lleons, per a poder implementar les zebres i per afegir elements com llacs i herbes/plantes per a què tinguin punts per beure aigua i menjar pels herbívors (veure Figura 14).

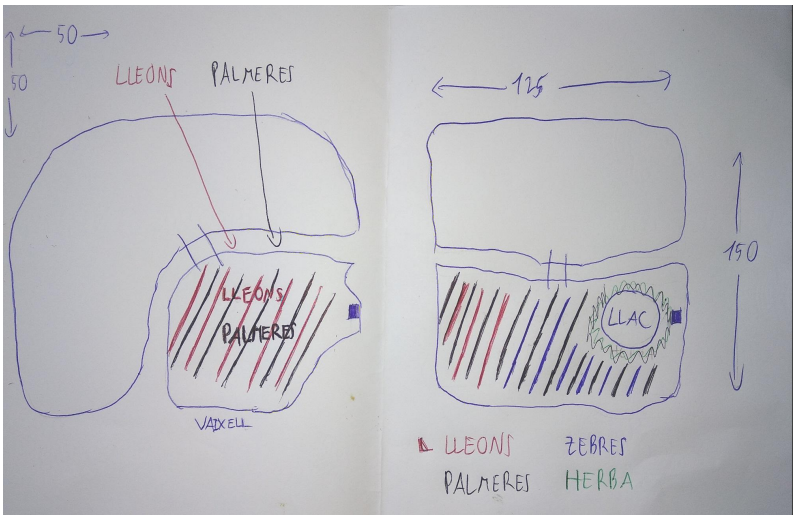


Figura 14: Esborrany del terreny de la Platja (inicial a la esquerra, futur a la dreta)

- **Bosc**: Les modificacions en aquest terreny són purament estètiques, és a dir, no cal ampliar el terreny, només afegir zones d'aigua dolça, dividir el terreny en tres parts per a poder implementar els conills i els pirates a banda dels llops, i posar una zona amb herba/plantes per a què els conills puguin menjar (veure Figura 15).

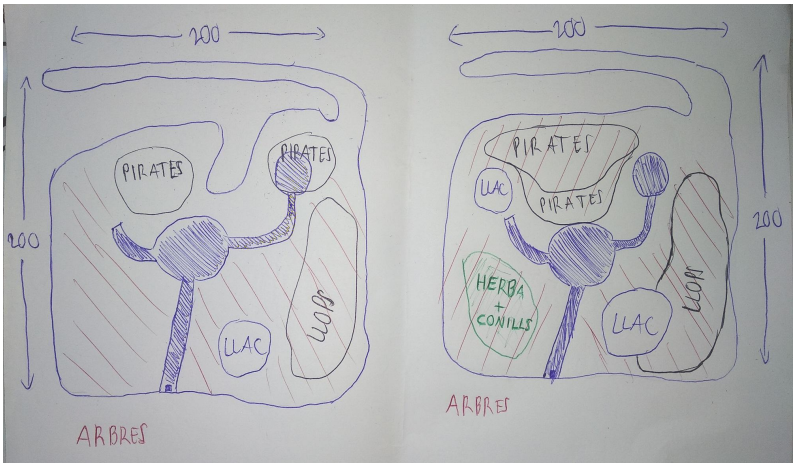


Figura 15: Esborrany del terreny del Bosc (inicial a la esquerra, futur a la dreta)

3.4 Requeriments tecnològics

A nivell de software, el joc no demana grans requisits ja que *Unity3D* és bastant general, només requereix que l'ordinador on s'executi tingui com a mínim el sistema operatiu Windows Vista en el cas de PC, Mac OS X en el cas d'un Mac o Ubuntu 12.04 en el cas de Linux.

A nivell de hardware, requereix una targeta gràfica que, com a mínim, tingui capacitat per a Dx10 (amb el model de shader 4.0), com per exemple la GS8400 de NVidia o la ATI Radeon HD 2400 XT per part d'AMD. També requereix una CPU compatible amb el sistema d'instrucions SSE2, com la Celeron de Intel o el Athlon 64 de AMD. En qualsevol cas, són components hardware prou antics com per no haver de suposar un problema, sobretot la CPU. La memòria RAM que el joc utilitza és d'uns 256MB aproximadament, així que amb un sol mòdul de 4GB n'hi ha prou. La memòria ROM que necessita el joc d'espai mínim és d'aproximadament mig GB. Ambdues quantitats són suficientment petites com per no haver de suposar un problema, ja que avui dia tots els ordinadors compleixen aquests requisists de memòria RAM i ROM.

Per altra banda, el joc requereix d'un servidor per tal de poder funcionar, el qual ha d'estar actiu en tot moment per a poder atendre les peticions d'informació del client com la informació de login, les fraccions de les missions del joc i poder gestionar la persistència del joc.

4 Disseny de la IA dels NPCs

Aquesta secció tracta del disseny de les diferents tècniques d'IA esmentades a la secció 2, que s'aplicarà als *NPCs* analitzats a la secció 3. D'aquesta forma s'obtindrà tot el disseny de la intel · ligència artificial aplicada al joc.

4.1 Moviment amb Navigation Mesh

La *Navigation Mesh* ha estat la tècnica més bàsica utilitzada pel control del moviment dels *Mobs*, ja que ha permès modelar les zones per on es podien desplaçar tots els personatges del joc, incloent el jugador, tal i com ha quedat esmentat en apartats anteriors.

El fet de poder modelar les zones disponibles pels personatges ha permès al mateix temps definir els límits desitjats per tal de no ser sobrepassats. En aquest cas, s'ha implementat amb l'objectiu de dessignar el límit de cada escena, és a dir, la part més propera al mar. També ha estat possible definir els límits a nivell intern, és a dir, deixar fora de la *Navigation Mesh* els diferents objectes del terreny com edificis, arbres i altres elements estàtics de vegetació, per evitar col · lisions amb aquests objectes.

4.2 Moviment amb Steering

Les diferents estratègies d'*Steering* també han estat utilitzades per a controlar el moviment de tots els *Mobs* amb excepció dels pirates, els quals han estat dotats d'altres tècniques com el BT, que s'encarreguen de controlar el moviment, entre d'altres aspectes de la seva IA. Tot i que el llop també implementa el BT per al control de la IA, utilitza una simple estratègia individual per assolir un punt.

Es poden dividir en estratègies de moviment individual i estratègies de moviment grupal.

4.2.1 Moviments bàsics de *Mobs*

Les estratègies d'*Steering* de moviment individual han estat aplicades a tots els *Mobs* (amb excepció dels pirates, com s'ha dit prèviament):

- La primera i més senzilla és la implementació dels ocells, basada en el ***Wander*** pur, què permet simular un moviment de vol completament aleatori, generant així un vol més realista que el que hi havia anteriorment.
  L'estratègia implementada en el TFG de partida es basava en el vol d'un punt a un altre escollit aleatòriament, mentre que l'estratègia actual calcula una direcció de forma aleatòria i interpola entre la posició en la que es troba i la futura utilitzant un factor d'interpolació que suavitza el moviment en diferents *frames*, és a dir, que no es mou en la nova direcció en el següent *frame*

sinó que assoleix la nova direcció en diferents *frames*. D'aquesta forma s'aconsegueix un moviment de gir suau, continu i aleatori en lloc d'un moviment completament rectilin i marcat, que és el que s'obtenia al anar d'un punt a un altre.
La diferència entre ambdues estratègies radica en la verosimilitud del moviment proporcionada per la nova estratègia.

- La segona estratègia individual ha estat aplicada a les zebres, als lleons i als conills, la qual es basa en juntar l'estratègia de moviment aleatori explicada en el paràgraf anterior (*Wander*) amb la de **lligar els mobs a una zona concreta** (*Tether*).
  L'objectiu d'unir-les és aconseguir que tant els lleons com les zebres apliquin el moviment aleatori i al mateix temps tinguin en compte l'àrea circular dins de la qual han de romandre. D'aquesta forma s'aconsegueix que mai surtin de la seva zona mentre es mouen aleatòriament.
  Cal afegir que aquesta estratègia no s'executa de forma completament lliure ja que simula un moviment constant dels *Mobs*, sense descans, fet que resulta del tot inverosímil ja que aquests animals (zebres, lleons i conills) acostumen a passar la major part del temps quiets, descansant. Per això s'ha implementat un control que restringeix el moviment aleatori a executar-se durant uns segons amb una probabilitat del 0.1%. L'script particular també va modificant la distància al Tether per evitar que facin ziga-zagues.

- Una altra estratègia utilitzada ha estat la d'**assolir un punt**, utilitzada per les zebres, els lleons i els conills a l'hora de beure aigua i només pels dos herbívors a l'hora de menjar.
  Quan la set supera un límit aleshores es desactiva l'estratègia de *Wander* + *Tether*, es fa una cerca del node d'aigua més proper al *Mob* i es posa com a objectiu. Després d'una estona d'haver-lo assolit, la necessitat d'aigua queda saciada, aleshores es desactiva aquesta estratègia i es torna a activar de nou la de *Wander* + *Tether*, fet que automàticament condueix al *Mob* cap a la zona en la què normalment viu. Si per casualitat dos *Mobs* molt propers van al mateix temps a beure aigua, el que arribi segon farà una cerca del següent node d'aigua més proper per tal de no topar amb l'altre *Mob*.
  El comportament amb la gana dels herbívors és idèntic, l'única diferència radica en què les plantes no són ïnfinitesçom els nodes d'aigua, és a dir, que per exemple un *Mob* menja la més propera (desapareixen al cap d'uns segons i tornen a aparèixer més tard) però no sacia per complet la gana, fet que l'obliga a buscar de nou la planta més propera i menjar-se-la. Repetirà el procés fins que la gana quedi saciada.
  L'últim *Mob* que l'utilitza és el pirata, amb l'objectiu d'assolir els punts establerts per les tècniques dels BTs compartits.

- Les dues últimes estratègies es basen en la de **perseguir**, implementada pels lleons, i la de **fugir**, implementada per les zebres i els conills. Ambdues estratègies també provoquen la desactivació de *Wander* + *Tether*.
  Quan la necessitat del lleó de menjar supera el límit aleshores s'activa l'es-

tratègia de perseguir les zebres. Aquseta estratègia provocarà que les zebres, quan tinguin un lleó aprop, fugin.

4.2.2 Moviments en grups: ocells, zebres

El control del moviment grupal ha estat implementat en ocells i zebres. Per tal de dur-lo a terme s'ha utilitzat una tècnica d'*Steering* basada en la combinació de tres components: alineació, cohesió i separació.

Aquesta tècnica consisteix en modelar una sèrie de paràmetres per tal de poder simular un moviment conjunt de una certa quantitat de *Mobs* al mateix temps, en la mateixa direcció, utilitzant l'alineació. Un cop modelat aquest moviment conjunt s'utilitzen les altres dues tècniques per tal de fer que els *Mobs* no se separin massa entre ells i evitar desmuntar el grup (cohesió) i per a evitar que, per altra banda, tots els *Mobs* col · lapsin uns a sobre dels altres, és a dir, per a donar un cert espai entre ells (separació). Amb tot això s'aconsegueix un moviment grupal fluid.

4.3 Comportament amb FSM

L'altre tècnica de IA aplicada al joc ha estat la màquina d'estats (FSM, *Finite-State Machine* en anglès), per tal de controlar els comportaments generals de certs *Mobs* del joc com els conills, les zebres i els lleons.

Degut a que les FSM han estat dissenyades amb l'objectiu de controlar el comportament en lloc de modelar el moviment, els dissenys del conill i la zebra són iguals, ja que ambdós fan el *Wander*, mengen el mateix i han de fugir dels seus respectius depredadors (veure Figura 16).

Al iniciar l'execució la màquina, comença pel *Wander* que és el que s'executa sempre per defecte si no s'ha d'executar cap altre. Primer es comprova si ha de fugir. Si és així, passa a l'estat de *Flee*, en cas contrari passa a comprovar les necessitats. Si té sed, passa a saciar-la fins que troba algun enemic (fet que provoca passar a l'estat de *Flee*). El mateix fa amb la gana. Quan ha passat el perill torna a *Wander*, on es torna a analitzar tots els atributs de nou.

La FSM del lleó té els mateixos estats i el mateix comportament que la FSM dels conills i les zebres, amb l'excepció de l'estat *Flee* ja que el lleó no té cap depredador i sempre ataca al jugador quan se'l troba (veure 17).

4.4 Comportament basat en Behavour Trees

Els BT han estat utilitzats en substitució de diverses estratègies d'*Steering* i les FSM, per tal de poder observar com actuaven els llops a l'hora de dur a terme les mateixes funcions que altres *Mobs* amb *Steering* i FSM (els lleons, concretament).

S'han considerat els BTs donat que modelen comportaments més complexes i, a més a més, s'han reutilitzat subarbres dels BTs pels diferents tipus de llops (*alpha* i *beta*).

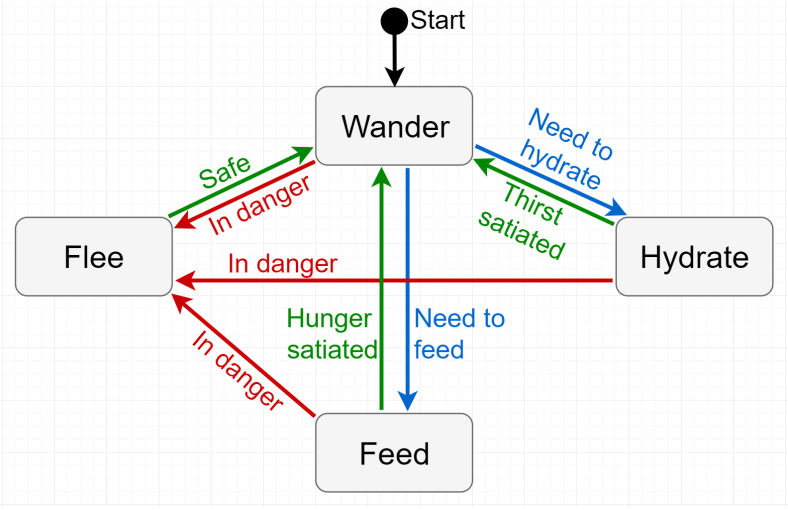


Figura 16: Disseny de la màquina d'estats del conill i la zebra.

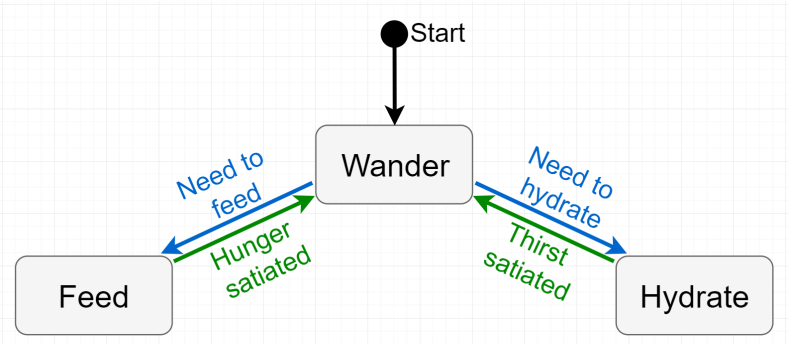


Figura 17: Disseny de la màquina d'estats del lleó.

4.4.1 Llops

Els llops ***beta*** han estat implementats amb un arbre de comportament format per cinc subarbres que surten del node arrel, que és de tipus selector. Els subarbres estan ordenats segons el grau d'importància del comportament, és a dir, si el primer subarbre (atac) es pot executar té prioritat sobre el segon (escapar). Si decideix que no ha d'atacar aleshores passa a comprovar si ha d'escapar, i així successivament. Dit BT només s'activa quan el llop és viu.

A continuació els subarbres del BT del llop:

- **Atacar**: Subarbre format per un node de tipus Seqüència que dóna lloc a 3 nodes fulla, equivalents a una seqüència d'accions que condueixen a atacar l'enemic (Figura 18):
  - Comprovar la distància: Node fulla que comprova si està suficientment aprop com per atacar. Si no ho està retorna *Failure* i passa al següent subarbre. Si es troba suficientment aprop de l'enemic com per haver d'atacar aleshores retornarà un *Success* i passarà al següent node fulla.
  - Comprovar els altres llops: Node fulla que comprova si hi ha llops aprop

per atacar i retorna *Success* si n'hi ha, passant així a l'últim node fulla de la seqüència. Aquest comportament es deu a que els llops no ataquen mai sols.

– Atacar al jugador: Últim node fulla que activa l'atac i comprova que el jugador és viu. Retornarà *Success* mentre el jugador segueixi viu i el llop pugui seguir atacant. En el cas, per exemple, que sigui el darrer llop atacant, ja no entraria a atacar perquè no hauria passat el node anterior.

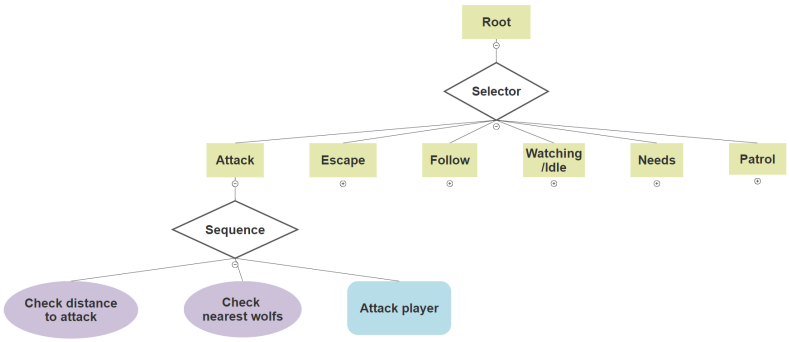


Figura 18: Disseny del subarbre d'atacar l'enemic.

- **Escapar**: Subarbre format per un node de seqüència que dóna lloc a 4 nodes fulla. Aquest subarbre és el que comprova la distància amb el jugador, i és el que sempre s'executa primer quan el jugador és aprop ja que els llops *beta* mai ataquen sols, sempre van a cercar un llop *alpha* o més llops *beta* (Figura 19):

  – Comprovar la distància: Node fulla que comprova la distància amb el jugador. Si està prou aprop com per fugir retornarà un *Success*.

  – Comprovar els llops *alpha*: Node fulla que busca el llop *alpha* més proper. Si el troba aprop aleshores no cal fugir sinó que pot atacar, i retorna un *Failure* per indicar que no cal fugir. En cas contrari vol dir que ha d'anar a buscar l'*alpha* ja que no es troba aprop, aleshores retorna un *Success* per indicar que cal seguir la seqüència del node pare.

  – Comprovar llops *beta*: Node fulla al qual passa si no troba cap *alpha* passa a buscar llops *beta*, ja que un mínim de dos *beta* tenen el mateix efecte que un *alpha*. En aquest cas retorna *Failure* per indicar que no cal fugir. En cas contrari retorna *Success*.

  – Escapar i trobar llops: Node fulla final de la seqüència. Si arriba a aquest node vol dir que no ha trobat cap *alpha* i/o menys de dos *beta*, per tant ha d'anar a buscar-los. Aleshores es fixa com objectiu el llop *alpha* més proper. Com que l'arbre s'avalua a cada iteració del joc no cal tenir en compte si arriba a aquest node, ja que constantment es comproven l'*alpha* i els *beta* propers al llop.

- **Perseguir**: Subarbre que s'executa quan el llop ha de perseguir al jugador (veure figura 20).

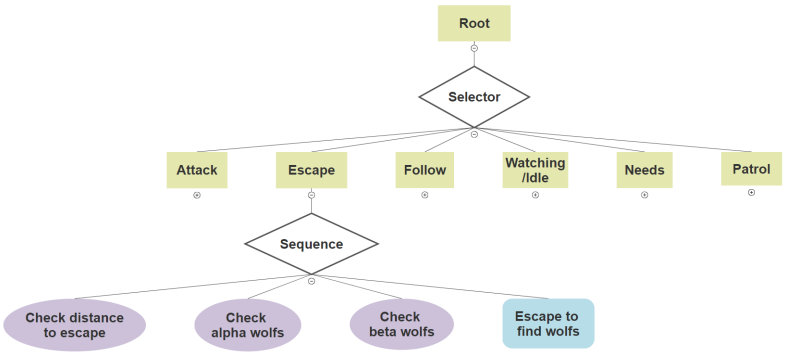


Figura 19: Disseny del subarbre d'escapar de l'enemic.

- Comprovar la distància: Primer node fulla de la seqüència. S'encarreqa de comprovar si està a la distància suficient com per a perseguir al jugador.
- Perseguir al jugador: Node fulla al que passa si es troba suficientment aprop com per perseguir-lo aleshores el fixa com objectiu i s'hi aproxima ràpidament. No cal controlar quan arribi a apropar-s'hi ja que quan estigui suficient aprop entrarà al subarbre d'atacar.

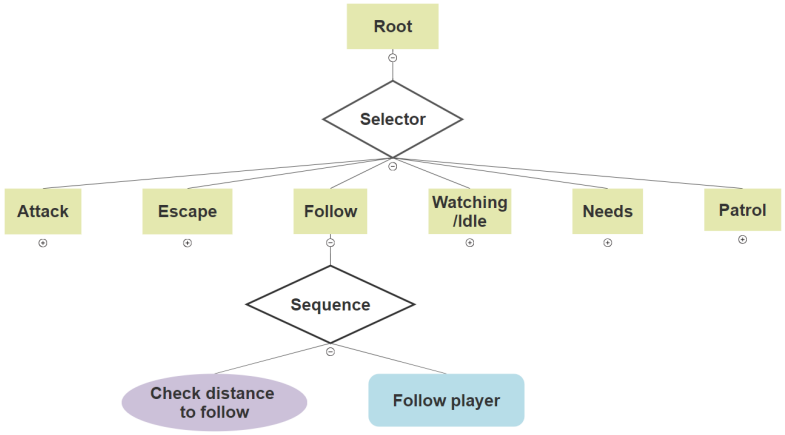


Figura 20: Disseny del subarbre de perseguir l'enemic.

- **Observar/estar quiet**: Subarbre que s'executa si l'enemic està prou lluny com per no haver de córrer cap enlloc ni atacar, però està prou aprop com per vigilar-lo aleshores entra en aquest subarbre. També es pot executar quan el *Mob* està prou cansat com per necessitar aturar-se (Figura 21).
  Està format per un selector que selecciona quina de les dues accions cal fer, sent prioritària la de vigilar. Es subdivideix en dues seqüències formades per dos nodes fulla cadascuna: el node de comprovació de la condició (*cal vigilar?* o *cal descansar?*) i el de la realització de l'acció corresponent.
  - Comprovar la distància: Node fulla encarregat de comprovar si hi ha alguna amenaça suficientment aprop com per simplement vigilar-la. Aleshores s'activa el booleà de vigilar i retorna *Success*.

- Vigilar l'enemic: Node fulla al què arriba si el booleà activat és el de vigilar, aleshores aquest node fulla s'encarrega de fixar la direcció del llop cap a l'amenaça.
- Comprovar el cansanci: Si està prou cansat i no hi ha cap amenaça aprop s'activarà aquesta condició i donarà pas al següent node fulla.
- Quedar-se quiet: Realitzarà l'acció de descansar (la mateixa que vigilar) fins que deixi de complir-se la condició. Quan deixa d'estar cansat torna a activar la velocitat i retorna un *Failure*.

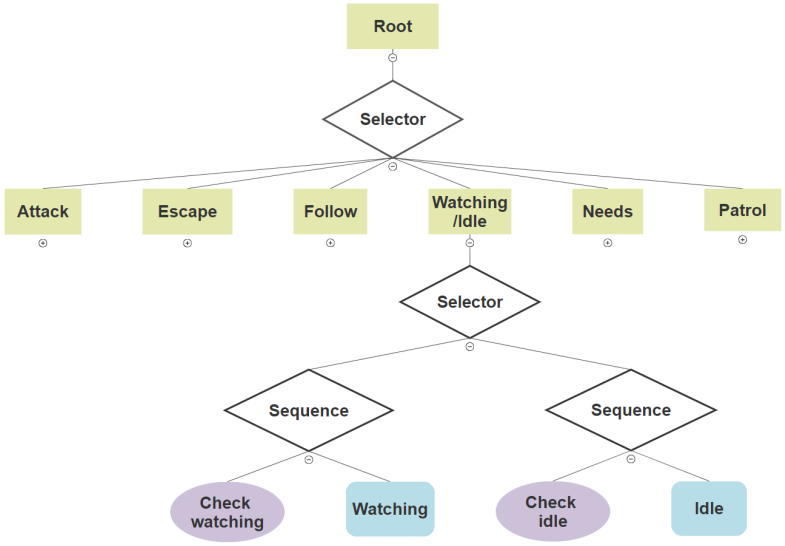


Figura 21: Disseny del subarbre de vigilar l'enemic o estar quiet.

- **Necessitats**: Just abans de poder patrulla el *Mob* comprova si cal executar el subarbre encarregat saciar alguna necessitat (aigua i/o menjar). Està format per un selector que selecciona la necessitat que cal ser saciada, sent la set la més important.
  Es divideix en dues seqüències. Ambdues formades per una condició que comprova si cal saciar la necessitat i per l'acció de saciar-la (veure figura 22):
  - Comprovar la set: Node fulla encarregat de comprovar si la condició de saciar la set s'acompleix. Si és així retornarà *Success*.
  - Saciar la set: Node fulla que busca el node del llac més proper i s'hi apropa. Quan l'ha trobat fixa el node central del llac (fa l'efecte de que beu amb el cap ajupit). Mentre tingui set retornarà un *Failure* ja que un *Success* indicaria que la set està saciada i pot passar a saciar la gana. Quan el booleà de la set estigui desactivat aleshores retornarà el *Success*.
  - Comprovar la gana: Si per altra banda s'acompleix la condició de saciar la gana aleshores entrarà al següent node fulla de la mateixa manera que la condició de la set.
  - Saciar la gana: Node fulla que té el mateix funcionament que el comportament dels lleons amb la diferència que menja conills. De la mateixa

manera que el lleó, a cada iteració busca el conill més proper i utilitza l'script de *SteerForPoint* per a fixar el seu objectiu fins que l'assoleix i el mata.

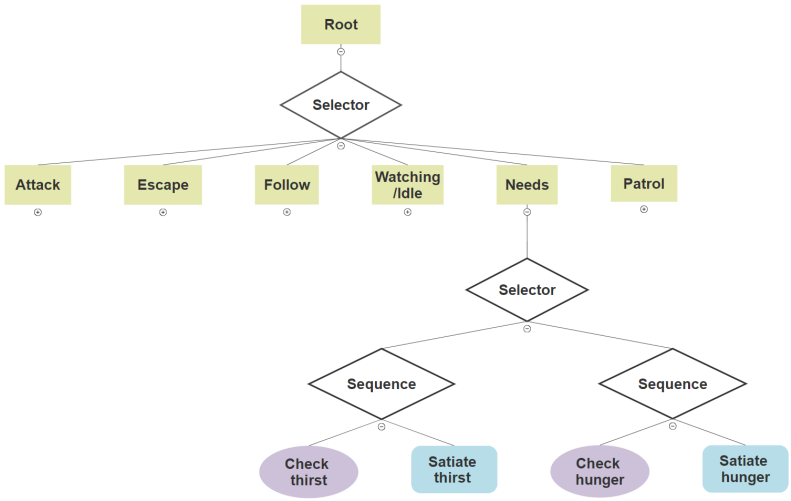


Figura 22: Disseny del subarbre de les necessitats.

- **Patrullar**: Quan no hi ha cap tipus d'amenaça a la vista aleshores el llop entra en aquest subarbre per tal de fer el moviment aleatori de *Wander* (Figura 23).
  - Comprovar si pot patrullar: Node fulla que comprova si algun booleà relacionat amb l'amenaça està activat. Només si no es compleix cap d'aquestes condicions segueix amb la seqüència.
  - moveToRandomPoint: L'últim node fulla del subarbre de patrullar és l'encarregat de seleccionar un node aleatori de patrullatge i mitjançant l'***SteerForPoint*** es mou en la direcció del node. Quan l'assoleix torna a buscar un altre node de patrullatge. Així indefinidament o fins que es trobi amb alguna amenaça.

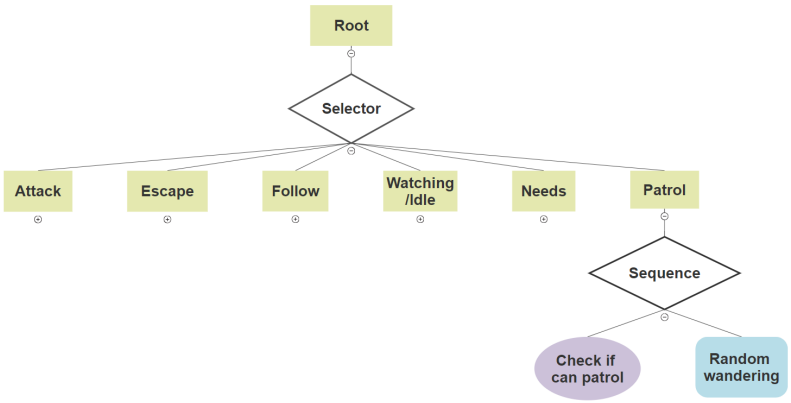


Figura 23: Disseny del subarbre de patrullar (moviment aleatori).

El llop ***alpha*** és un tipus de llop que no fuig mai del perill, sinó que sempre enfronta l'amenaça. El subarbre utilitzat ha estat el mateix en ambdós tipus de llops, però amb la diferència que l'*alpha* no té el subarbre d'escapar (veure figura 24). Gràcies a la possible reutilització de subarbres dels BTs, el disseny del llop *alpha* pot dissenyar-se de forma fàcil, sense haver de dissenyar nous arbres de comportament per atacar, perseguir, patrullar, etc.

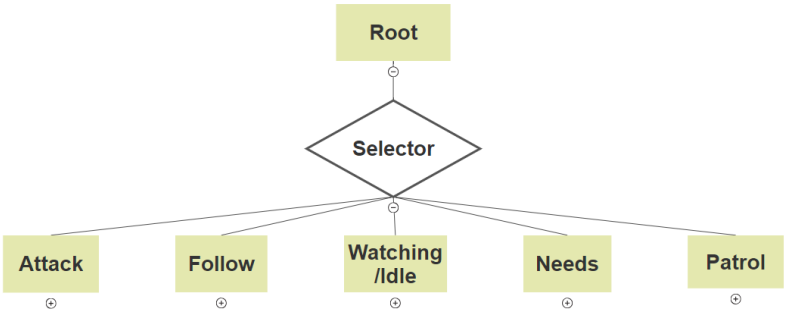


Figura 24: Disseny del arbre d'un llop *alpha*.

4.5 Comportament basat an BTs compartits: Pirates

Finalment, l'última estratègia utilitzada ha estat la dels BTs amb una mèmoria compartida o *blackboard*, dissenyada per a sincronitzar una part dels BTs per simular formacions de diferents pirates.

4.5.1 Pirates1: BTs sincronitzats

Aquests BTs són molt semblants als dels llops però amb la diferència que els pirates tenen un subarbre encarregat d'executar tota la part de formació de grups i de sincronització de BTs mitjançant la *blackboard*. Aquest subarbre és el que s'executarà primer en el recorregut del BT (veure figura 25).

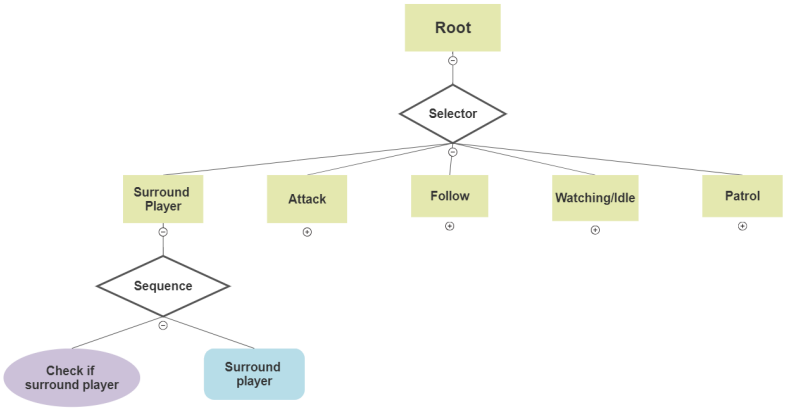


Figura 25: Disseny del subarbre d'envoltar el jugador.

El subarbre està dividit en dues parts. La primera és la condició que s'encarrega de comprovar dins la *blackboard* si algun pirata ha activat la movilització per agrupar-se. La segona part és l'acció següent a la condició, on el pirata accedeix a la *blackboard* per a obtenir la posició que li correspongui (veure figura 26).

En el cas que no estigui activa, i el pirata és proper al jugador, l'activa.

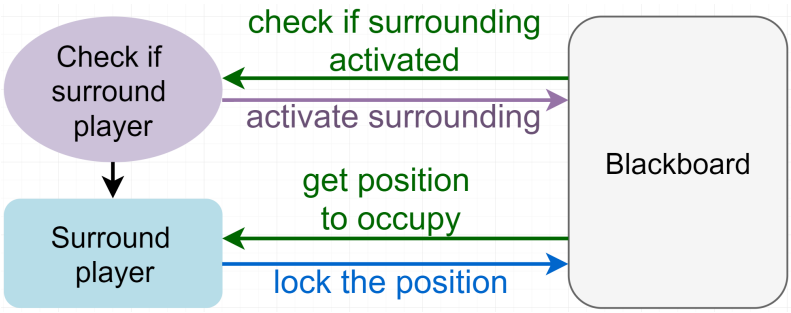


Figura 26: Fluxe d'informació entre el subarbre i la *blackboard*.

4.5.2 Pirates2: Formacions

Els pirates duran a terme una formació per a rodejar el jugador mitjançant la sincronització dels seus respectius BTs.

Quan un pirata està en el rang de visió del jugador activa el moviment sincronitzat a la blackboard, el que indica que s'han de posar tots els pirates en formació. Fet això, aquest primer pirata es queda vigilant al jugador (veure Figura 27a).

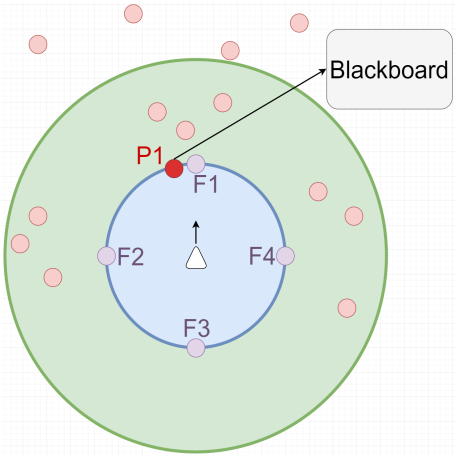
A continuació els demés pirates comprovaran si cal sincronitzar moviments i cadascun demanarà una posició (veure Figura 27b). Segons on es trobi en aquell moment cada pirata se li assignarà un flanc concret. La posició dins del flanc depèn de la quantitat de pirates que ja estan en aquell flanc. Sempre s'assigna la següent posició lliure (veure Figura 27c).

Finalment, el jugador queda envoltat pels pirates que es trobaven dintre del rang de color verd, com es pot veure a la Figura 27d).

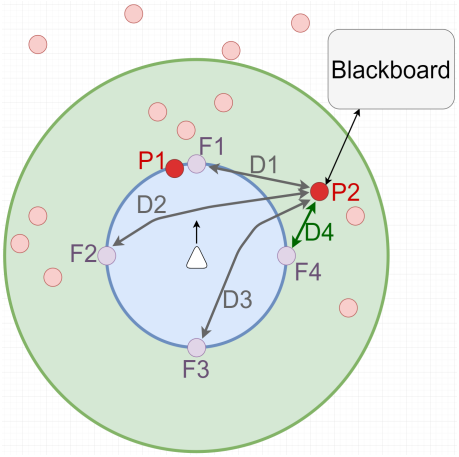
De la mateixa manera que amb el primer pirata, quan un altre pirata acudeix a la crida del primer i assoleix la posició que li toca, aleshores passa a vigilar el jugador també, sense tornar a executar el subarbre d'envoltar.

En la ***blackboard*** es guarda una llista de flancs que envolten al jugador, quatre per defecte, i una llista d'objectes que guarda la informació relativa als pirates que ocupen posicions en cada flanc (per defecte cada flanc està dividit en vint posicions). D'aquesta forma cada cop que un pirata s'uneix a la formació se li assigna una posició en el flanc més proper, tenint en compte tots els llocs ocupats en aquell flanc, és a dir, que si el flanc més proper té dos pirates aleshores es calcularà la tercera posició i se li assignarà. Finalment, el pirata serà afegit a la subllista corresponent al flanc on estigui posicionat, emmagatzemada a la *blackboard*.

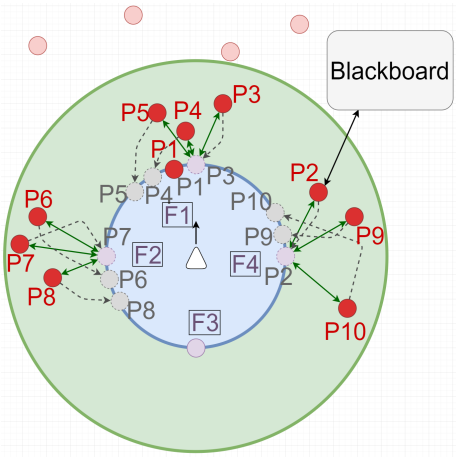
A cada iteració del joc només cal actualitzar les posicions dels flancs respecte



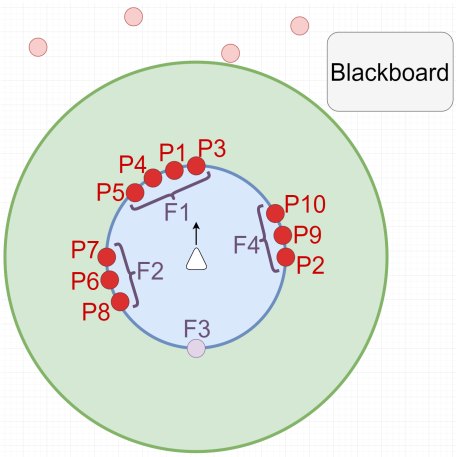
(a) Primer pirata *P1* (vermell) que veu el jugador i activa la sincronització grupal.



(b) Pirata *P2* (vermell) que decideix el flanc segons la distància *D1*-*D4* amb tots ells, quedant-se amb el més proper. *D4* (fletxa verda) és la més petita, per tant *F4* és el flanc escollit.



(c) Flanc més proper per cada pirata (fletxa verda, per exemple de *P1* a *F1*) i la posició que ocupa dintre del flanc (fletxa gris de punts).



(d) Posicions finals de cada pirata dintre dels flancs que els corresponien per proximitat.

Figura 27: Disseny de la sincronització de BTs dels pirates per a obtenir la posició. El triangle blanc és el jugador, els punts vermells són els pirates, els liles són les posicions dels flancs, l'àrea circular blava és la distància a la qual els pirates veuen el jugador i l'àrea circular verda és la distància d'envoltar el jugador.

el jugador, i, recorrent la llista de la *blackboard*, s'actualitzaran les posicions dels pirates que estan ocupant els flancs.

5 Desenvolupament del joc

Per últim, aquesta secció explica les diferents eines que proporciona *Unity3D* per a poder implementar en el joc les tècniques d'IA dissenyades a la secció anterior. També exposa tota la modificació que ha sofert el joc des de la versió inicial junt amb el diagrama de totes les classes modificades i implementades. Finalment mostra les estratègies especials utilitzades al joc que han servit d'ajuda per a implementar correctament les tècniques d'IA.

5.1 Justificació de la tecnologia utilitzada

Per tal de desenvolupar, el joc s'ha utilitzat l'engine gràfic *Unity3D* [6], tal i com feia l'anterior versió del joc. El fet que *Unity3D* és un dels engines gràfics lliures més utilitzats, amb una gran quantitat de documentació i especialment orientat als jocs 2D, 2.5D i 3D senzills, com Fracsland, ha sigut decisiu per a continuar usant el mateix engine o motor gràfic.

Unity3D també compte amb una gran quantitat de material gratuït amb el què complementar els projectes, ja sigui completament de lliure distribució com els projectes de github [5] i altres repositoris online [4], o els assets gratuïts de l'Asset store de *Unity3D*, botiga online que conté software emmagatzemat en paquets (assets) desenvolupat especialment per a ser utilitzat en *Unity3D*.

5.1.1 *Navigation Mesh* de Unity3D

La ***Navigation Mesh*** de *Unity3D* és un software incorporat a l'engine gràfic base de *Unity3D* [3]. S'ha utilitzat perquè és relativament senzill d'utilitzar i, al mateix temps, fa possible dissenyar diverses formes de controlar la zona per la qual es mouen tots els personatges del joc, ja que permet definir diferents capes de navegació amb cost per tal de modelar el *pathfinding* (veure Figura 28.

Figura 28: Exemple de diferents capes amb els seus respectius costos d'una *Navigation Mesh*.

El *pathfinding* és una estratègia usada quant un NPC s'ha de moure. Amb aquesta estratègia es calcularà el millor camí buscant sempre el cost més baix).

La *Mavigation Mesh* també es fa servir per a restringir el moviment de certs NPCs a certes capes. Si hi ha NPCs terrestres com zebres i NPCs aquàtics com peixos es poden definir dues capes, una terrestre i una altra a l'aigua per tal d'evitar que

les zebres entrin a l'aigua i que els peixos vagin al sòl. Les capes es poden modelar mitjançant els següents paràmetres:

- **Inclinació màxima**: Tot el terreny inclinat amb un angle inferior al determinat pel paràmetre quedarà cobert per la capa.
- **Alçada màxima d'esglaó**: Per a evitar que qualsevol petit esglaó (de 90 graus aproximadament, que quedaria fora de la inclinació màxima) talli la capa
- **Dimensions de la capa vers el NPC**: El radi defineix la distància mínima que ha d'haver-hi entre el límit de la capa i el límit del terreny (radi=2 indica que la capa estarà sempre a dues unitats de distància del límit, que serà qualsevol part del terreny amb una inclinació major a la màxima permesa o amb un esglaó més alt que el màxim permès, com una pendent massa inclinada, una paret o un precipici).

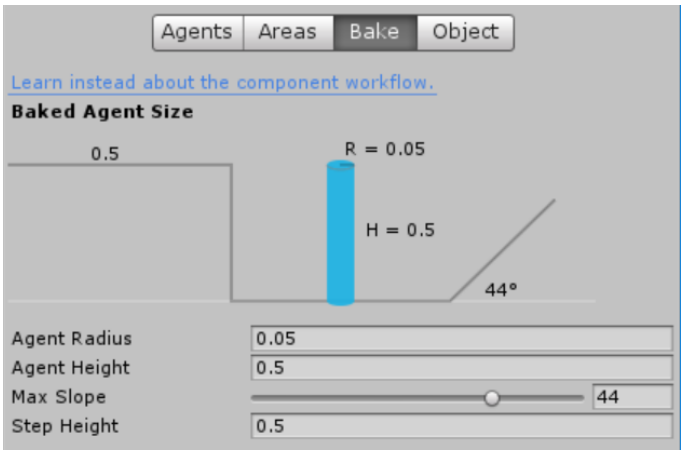


Figura 29: Exemple de paràmetres de la *Navigation Mesh*.

Cada NPC que implementa el component de la *Navigation Mesh* és un *Nav Mesh Agent* i pot instanciar els mateixos paràmetres de la *Navigation Mesh* general, però aplicats concretament al NPC. La inclinació màxima defineix el pendent màxim pel qual es pot moure el NPC; l'alçada màxima d'esglaó indica l'alçada que es podrà ignorar; i les dimensions del NPC indiquen l'àrea que ocupa el NPC (radi) i l'alçada. Aquestes dades defineixen un cilindre que serà tingut en compte per la capa de la *Navigation Mesh* quan el NPC arribi a un límit de la capa o hagi de passar per sota d'algun "sostre"massa petit (veure Figura 30).

5.1.2 UnitySteer

UnitySteer [5] és un projecte de github que proveeix d'una sèrie de scripts que permeten implementar tota una sèrie d'estratègies d'*Steering*. S'ha utilitzat aquest software ja que està molt extès, té una comunitat activa que l'actualitza i és senzill

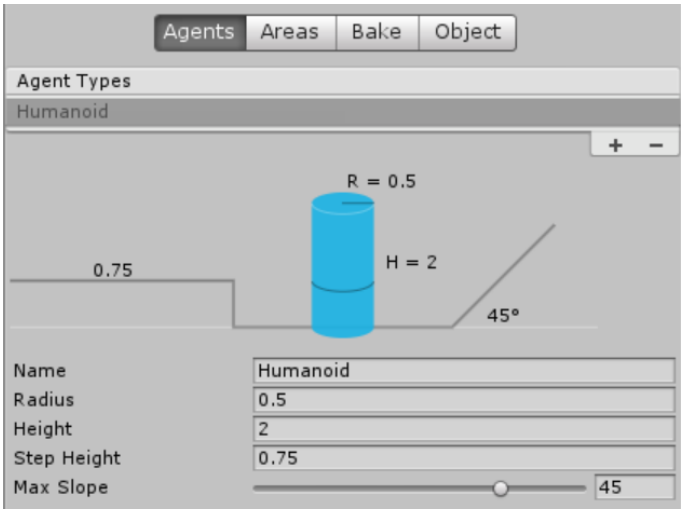


Figura 30: Exemple de paràmetres d'un NPC de la *Navigation Mesh*.

d'intregrar en un projecte. Els scripts d'*Steering* implementen una sèrie de paràmetres que permeten modelar el comportament desitjat de cada tècnica i, per tant, definir el comportament global de les estratègies implementades.

Incorpora scripts que permeten implementar les estratègies i tècniques de *steering* introduïdes en la secció 2.2.2 i es resumeixen a la figura 31.

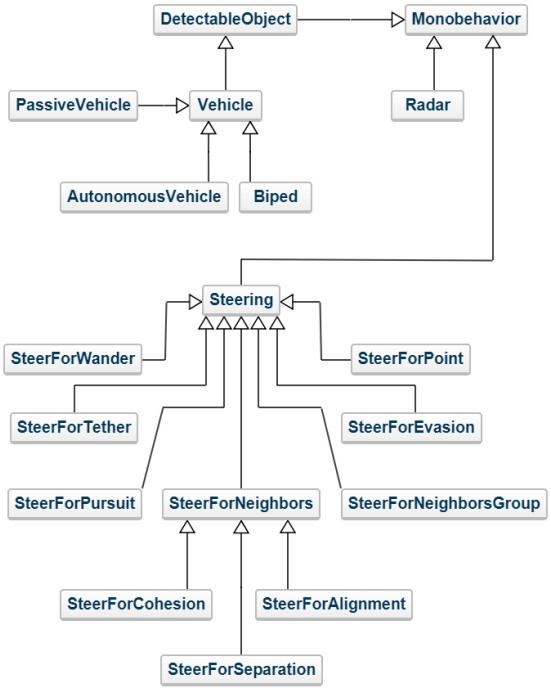


Figura 31: Diagrama de les classes de *UnitySteer* utilitzades.

A continuació es resumeixen les seves característiques específiques que permeten la seva integració en un projecte de *Unity3D*. S'han agrupat en tres tipus de scripts: els que s'encarreguen d'activar tots els altres scripts de moviment, scripits que

modelen comportament individual i els que modelen comportament grupal:

- Els **scripts d'activació**, encarregats d'activar tots els altres scripts de moviment. Qualsevol *Mob* que incorpori *UnitySteer* necessita un dels tres scripts descrits a continuació, ja que són els que recorren tots els components d'*Steering* del *Mob*, executant el càlcul dels seus respectius vectors de moviment i obtenint el vector final. Utilitzen els vectors de cada *Mob* per a moure'l:
  - ***PassiveVehicle***: Script aplicat als *Mobs* que no s'han de moure.
  - ***AutonomousVehicle***: Script aplicat als *Mobs* que es mouen només cap endavant. Utiliten el vector *FORWARD*, que defineix l'eix *Z* dins dels eixos de coordenades locals del *Mob*, és a dir, l'eix que en defineix la direcció ja que sempre apunta cap endavant (veure Figura 32).
  - ***Biped***: Script aplicat als *Mobs* que es poden moure en ambdues direccions, cap endavant i lateralment, definits pels vectors *FORWARD* i *RIGHT* respectivament. *RIGHT* és el vector que defineix l'eix *X* dins els eixos de coordenades locals del *Mob*. Amb això s'aconsegueix un moviment més propi dels bípeds, capaços de moure's en totes direccions sense haver de girar (veure Figura 32).
- **Script de *Radar***: Donat un radi de detecció, permet al *Mob* portador detectar tots els altres *Mobs* que tinguin un dels scripts anteriors, degut a que aquests hereten de la classe *DetectableObject*

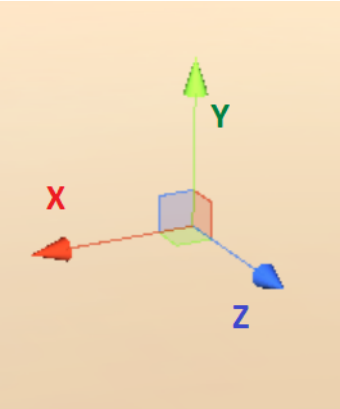


Figura 32: Eixos de coordenades de qualsevol objecte de *Unity3D*. El blau (Z) equival al *FORWARD* (vector de la direcció de l'objecte), el vermell (X) equival al *RIGHT* i el verd (Y) equival al *UP*.

- Scripts de **modelatge individual** del moviment dels *Mobs* són els següents:
  - ***SteerForPoint***: Donat un punt tridimensional calcula el vector per anar en la direcció adequada per a arribar fins a aquell punt.
  - ***SteerForTether***: Donats un punt tridimensional i un radi permet definir una àrea en la qual el NPC ha d'estar sempre. Si no hi és, i, depenent del pes, es calcula el vector de direcció que indica el camí que ha de prendre el *Mob* per a tornar dintre l'àrea.

- ***SteerForWander***: Donats uns límits pels vectors *UP* i *RIGHT* i un *smoothRate* per a suavitzar el moviment del *Mob* permet definir un cert comportament de moviment aleatori. El vector *UP* és el vector *Y* dins dels eixos de coordenades locals del *Mob*, és a dir, el vector d'alçada. (veure Figura 32)
- ***SteerForEvasion***: Donat un *Mob* considerat com a amenaça i una distància de seguretat, permet fer que el *Mob* principal comprovi si el *Mob* amenaça està prou lluny com per a haver de fugir. Un cop superada aquesta distància el càlcul del vector director final considera fortament l'amenaça i fa que dit vector sigui en direcció contrària a l'amenaça. També té un paràmetre de predicció de temps (*predictionTime*) que permet fer aquest càlcul del vector predint on estarà l'amenaça en un temps determinat.
- ***SteerForPursuit***: Té el mateix funcionament que l'anterior, amb l'única diferència que el vector calculat apunta directament al *Mob* objectiu. En aquest cas el paràmetre de la distància (*acceptableDistance*) té en compte la distància mínima a la que ja pot aturar la persecució ja que està prou aprop.

- Scripts de **modelatge grupal** del comportament dels *Mobs*:
  - ***SteerForNeighbors***: Script del qual hereten els scripts de comportament grupal (els tres scripts explicats a continuació). Incorpora dues distàncies (mínima i màxima) utilitzades pels scripts base per a determinar els radis mínim i màxim d'actuació, és a dir, que només tindrà en compte els altres *Mobs* que es trobin entre la distància mínima i la màxima. Els comportaments globals són:
    - ***SteerForCohesion***: Donat un pes calcula el vector necessari per a apropar-se més als *Mobs* que trobi al voltant (veure Figura 10a de la secció 2.2.3).
    - ***SteerForSeparation***: Donat un pes calcula i una distància de comfort (entre la màxima i la mínima) calcula el vector necessari en direcció al punt més adient amb l'objectiu de mantenir una distància de separació amb els *Mobs* que el rodejen tant propera com sigui possible a la distància de comfort dessignada (veure Figura 10b de la secció 2.2.3).
    - ***SteerForAlignment***: Donat un pes calcula el vector necessari per a seguir en la mateixa direcció que tots els *Mobs* propers (veure Figura 10c de la secció 2.2.3).
  - ***SteerForNeighborsGroup***: Script que s'encarrega de trobar tots els scripts implementats al *Mob* que heretin de SteerForNeighbors i calcular el vector de direcció de cadascun d'ells per tal d'obtenir finalment un sol vector (ponderat dels altres) que indiqui la direcció final grupal. Per a calcular aquest vector, té un paràmetre important a banda del pes, el camp de visió (en graus), que permet indicar la visió del *Mob* partint

del vector FORWARD fins a un màxim de 180 graus, que equivaldria un camp de visió complet (veure Figura 33).

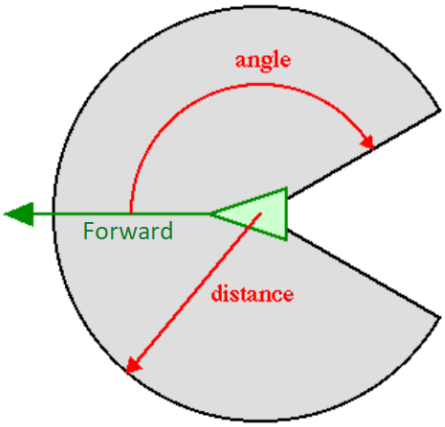


Figura 33: Camp de visió del *Mob* per detectar altres *Mobs*. El *Mob* és el triangle verd, junt amb el seu vector *FORWARD* que defineix la direcció (veure Figura 32).

5.1.3 Projecte FSM

FSM és un projecte de lliure distribució que proveeix d'una interfaç per poder implementar màquines d'estats finits sense haver de programar la lògica interna d'una màquina d'estats [4].

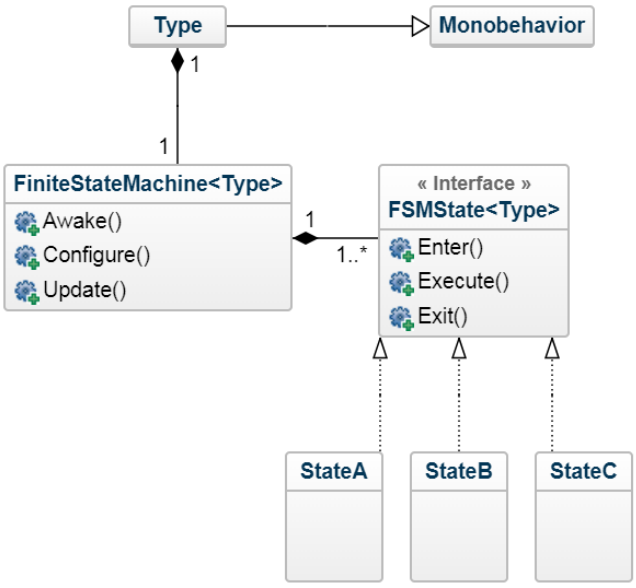


Figura 34: Diagrama de les classes de FSM

Conceptualment, cada estat dins de la màquina ha de ser una classe, i per això proveeix d'una interfície (*FSMState<T>*, on T representa el *Mob*) de la qual han

d'heretar totes les classes que hagin d'implementar un estat diferent. Aquesta interfície implementa tres mètodes, el d'entrada a l'estat, el que s'executarà mentre l'estat estigui actiu i el de sortida cap a un altre estat.

Funciona de forma que cal instanciar-la amb la classe que l'utilitzarà (com pot ser *Zebra*: *FiniteStateMachine<Zebra> fsm*) i executar els mètodes d'inicialització *fsm.Awake* (inicialitza els atributs amb valors nuls) i *fsm.Configure* (semblant al mètode *Start*, activa la màquina amb l'estat passat per paràmetre) per tal d'engegar-la.
Un cop definits aquests passos, només cal cridar el mètode que actualitza la màquina d'estats (*fsm.Update*) cada cop que sigui necessari.

En cridar el mètode d'actualització de la màquina d'estats, aquest cridarà el mètode d'execució de l'estat actiu, el qual ha d'implementar una lògica que determini si cal canviar d'estat o no. En cas afirmatiu, és necessari cridar el mètode *fsm.ChangeState* per indicar a la FSM l'estat que estarà actiu a partir de la següent execució del mètode d'actualització.

5.1.4 PandaBT

PandaBT és un asset gratuit de l'asset store de *Unity3D* que permet utilitzar BTs sense necessitat d'haver d'implementar tota la lògica interna d'un arbre de comportament [2].

PandaBT divideix en dues parts el disseny de l'arbre per tal de separar la lògica de IA que implementa el programador amb la lògica interna. En primer lloc proveeix de la comanda ***[Task]*** que permet dessignar els mètodes que pertanyen a la IA del BT, és a dir, els que equivaldran als nodes fulla de l'arbre.

En segon lloc, PandaBT permet dissenyar l'arbre utilitzant un simple archiu *.txt*. El programador pot implementar tota l'estructura de branques internes i nodes fulla del BT mitjançant les estructures que defineix PandaBT. Tot i que n'hi ha més, les següents són les que han estat utilitzades:

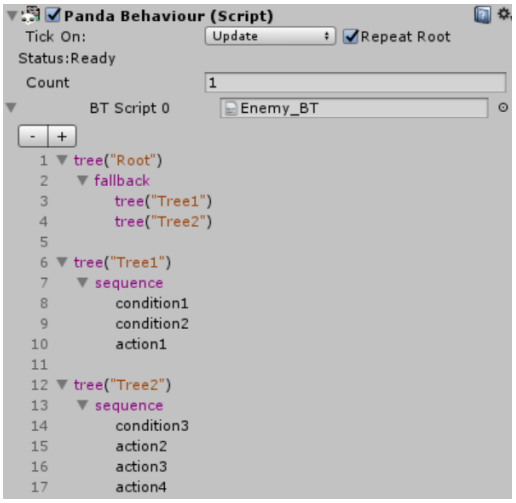
- ***tree("Name")***: Permet definir un subarbre que pot ser implementat a banda de l'arbre principal, com si d'un mètode es tractés ja que pot ser referenciat posant *tree("Name")*. Qualsevol subarbre té un fill. Sempre cal definir com a mínim l'arrel (*tree("Root")*) del qual partirà l'execució de l'arbre. Normalment s'utilitzen per a implementar tots els comportaments de IA que pugui tenir el NPC.

- ***fallback***: Usualment conegut com a *selector*, permet implementar una seqüència que itera sobre els fills, executant-los i fallant (retornant *Failure*), fins que executa un que retorna *Success*, aleshores tornarà a començar pel *tree("Root")* el següent cop que iteri sobre l'arbre. Permet tenir tants nodes fills com el programador vulgui, que acostumen a ser subarbres.

- ***sequence***: Permet implementar una seqüència que funciona inversament al *fallback* ja que itera sobre els nodes fills fins que falla un, aleshores retorna

Failure i acaba la seqüència. Només retorna *Success* quan ha iterat sobre tots els nodes fills i tots han retornat *Success*. Permet tenir tants nodes fills com el programador vulgui, que acostumen a ser nodes fulla.

- **Node fulla**: Nodes amb el nom dels mètodes que s'han d'executar (condicions i accions), implementats en un script que s'ha d'incloure com a *Component* al *GameObject*Tot i que no és cap branca o node intern de l'arbre, cal posar-ho ja que són els mètodes de IA escrits en un o més scripts, amb la comanda *[Task]* a la capçalera del mètode (veure Figura 35).



(a) Exemple d'una implementació de BT amb PandaBT.

```
[Task]
public void condition1()
{
    if (Random.value < 1)
        Task.current.Succeed(); //Success
    else
        Task.current.Fail();    //Failure
}
```

(b) Exemple de la implementació en un script d'un node fulla del BT.

Figura 35: Exemple de les dues parts del *PandaBT*, el component de *Unity3D* (esquerra) i un mètode dels que executa com node fulla (dreta).

5.2 Modificació del joc inicial Fracsland

La versió inicial del joc depenia d'un servidor implementat en *Django* [1], el qual permetia la persistència de les dades dels jugadors (usuari i contrasenya, classe i curs a la que pertany l'usuari, partides fetes, etc.) i també la interacció amb el joc en el moment de l'execució per tal de controlar l'avanç de l'usuari per així poder enviar i rebre tota la informació necessària, com per exemple les fraccions que ha de fer el jugador segons el nivell.

Donat a que el servidor remot en el què estava connectat el joc on estava instal · lat el servidor de Django donava problemes de connexió i enlentia el funcionament del joc, s'ha decidit desconnectar el joc del servidor. S'ha partit de les dues implementacions de testeig de servidor ja integrat en Fracsland, una amb el remot i una altra en local. La implementació local tenia un usuari i contrasenya per defecte i molt poques peticions d'informació amb el servidor. Això ha permès desenvolupar un senzill servidor en local com a proxy que atén les peticions del joc i permet executar-lo. A més a més ha estat possible l'execució del joc en local ja que no

calia implementar cap tipus de connexió amb el servidor, que els únics canvis que s'havien de dur a terme eren a nivell de IA, amb els diferents NPCs.

Un cop solucionat aquest problema i podent probar el joc, també es van trobar diversos errors o *bugs* que calia solventar. Primer de tot, quan el jugador era assassinat per qualsevol *Mob*, el personatge del jugador apareixia al començament del mapa, però el *Mob* el perseguia constantment. Això era degut a que el *Mob* no ressetejava els seus atributs al matar el jugador. Similarment, quan el jugador matava un *Mob* no deixava de colpejar-lo, encara que el *Mob* ja hagués desaparegut.

Un altre problema que hi havia està relacionat amb el *transform.lookAt*, mètode de qualsevol personatge que permet fixar el vector *forward* cap a un punt tridimensional concret. El problema sorgia quan s'utilitzava aquest mètode per a focalitzar un element que no estava a la mateixa alçada que el personatge, ja que aleshores aquest es podia tombar cap amunt. Només calia focalitzara cap a un punt bidimensional, utilitzant sempre l'alçada del personatge. Per últim, un problema que va propiciar l'ús de la *Navigation Mesh* fou que qualsevol personatge podia travessar els límits, com per exemple a la platja, on el jugador podia ficar-se dins el mar, o per exemple travessar objectes com edificis o elements de la vegetació.

Altres petites millores que s'han realitzat són les que milloren l'experiència del joc, com per exemple, permetre que la càmera principal es traslladi utilitzant el botó dret del ratolí, enfocant sempre el jugador, en lloc de deixar-la estàtica com era abans. Una millora a la interfície gràfica del joc ha estat implementar el control de la tecla *M* per a poder fer desaparèixer el mapa quan no es necessita, ja que ocupa molt espai. Una millora de les animacions dels mobs ha estat implementar dos *Animators* per tal de no haver de controlar les animacions per codi i per a poder fer transicions suaus entre animacions per simular suavitat de moviments, com per exemple entre estar quiet i córrer (en lloc d'iniciar l'animació de córrer inmediatament es fa una transició que simula l'acceleració de començar a córrer. També s'han implementat altres animacions com la de menjar quan un *Mob* estigui menjant.

5.3 Diagrama de classes en Unity3D

[12]

Totes les estratègies d'*Steering* implementades no funcionen completament alienes a l'script particular afegit a cada *Mob*. Depenen d'ell en aspectes com l'estratègia de punt (activada per l'script particular del *Mob* que li dóna un punt objectiu i un pes) o la de fugir/perseguir (activada també per l'script que porporciona el *Mob* predador o presa, respectivament).

Cal notar que tots els *Mobs* tenen l'script base d'*AutonomousVehicle*, i els que utilitzen l'estratègia grupal tenen l'script *Radar* (ocells i zebres).

En aquest apartat queden detallats els diagrames de les classes que formen la IA de cada *Mob* implementat. Tots ells tenen el *Mob (GameObject)* com a centre del diagrama, ja que és el que reuneix els scripts d'*Steering*, els particulars de cada

Mob, les FSM i els BT. En primer lloc es visualitzen els de la platja:

Els ocells utilitzen pur *Steering*, ja que no tenen cap script particular que els controli. Utilitzen una simple estratègia de moviment aleatori grupal, composta per la classe *SteerForWander* per a modelar el moviment aleatori d'un sol ocell i les classes de moviment grupal *SteerForCohesion*, *SteerForSeparation* i *SteerForAlignment*. En juntar aquestes dues estratègies s'ha aconseguit simular el vol grupal aleatori.

Per tal d'evitar que els ocells col · lapsin els uns sobre els altres ha calgut calibrar els paràmetres de l'estratègia grupal i donar més pes a l'script de separació que al de cohesió, ja que sinó finalment quedaven totalment enganxats. En contrapartida, aquesta diferència entre el pes de separació i el de cohesió provoca que poc a poc es formin grups d'ocells cada cop més llunyans entre ells. Això és degut a que el càlcul del vector grupal pondera més sobre els *Mobs* propers, fet que provoca que els *Mobs* més llunyans es tinguin molt poc en compte, prenent direccions diferents, fins que arriba un moment en què són totalment aliens els uns amb els altres.

Tot i així l'script amb més pes ha estat el d'alineació. Com els ocells estan en constant moviment en volar doncs fàcilment prenien direccions diferents, quedant separats els uns dels altres molt ràpidament. Donar més pes a l'script *SteerForAlignment* ha permès que els ocells tinguessin prioritat per seguir la direcció del grup, fet que ha alentit el procés de divisió d'un grup d'ocells en altres més petits.La figura 36 detalla els components dels ocells, que en aquest cas només són els scripts d'*Steering*.

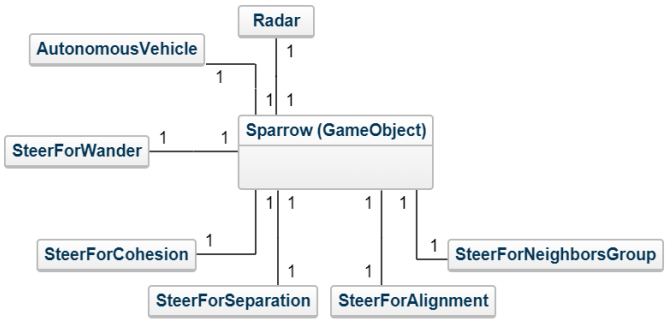


Figura 36: Diagrama de les classes del *Mob Sparrow*

Tots els altres *Mobs* utilitzen diverses estratègies d'*Steering*:

- **Estratègia grupal**: Aplicada només a les zebres. Ha estat necessari fer una distribució de pesos semblant a la utilitzada amb els ocells, ja que per defecte les zebres també acabaven col · lapsant les unes sobre les altres, tot i que la diferència de pesos no és tan gran com la dels ocells, i concretament el pes de l'alineació és el mateix que el de la separació. Això és degut a que el *Wander* de les zebres no és constant i és restringit a una zona.

- **Estratègia de zona**: Aplicada a les zebres, els lleons i els conills. Utilitza el *SteerForWander* per a simular el moviment aleatori i el *SteerForTether* per a

definir el centre de la zona en la que han de romandre els *Mobs*. De la mateixa forma que els ocells, aquesta estratègia també ha quedat unida a la grupal per tal de simular el wander en grup en el cas de les zebres.

- **Estratègia de punt (objectiu)**: Aplicada a tots els *Mobs* amb excepció dels ocells. Fent ús del *SteerForPoint* s'ha implementat l'estratègia per a fer que els *Mobs* puguin dirigir-se a cada objectiu particular, com els nodes dels extrems del llac per beure aigua (tots els mobs), les plantes que mengen els herbívors (zebres i conills) o els nodes de moviment que implementen els *Mobs* que no implementen l'estratègia de *Wander* (llops i pirates. El pes i el punt d'aquest script es controlen desde l'script particular.

- **Estratègia de fugir**: Aplicada als conills i zebres. Mitjançant els paràmetres de l'script *SteerForEvasion* es pot determinar l'amenaça de la qual cal fugir (un *GameObject* que incorpori un dels tres scripts que hereten de *Vehicle*) i la distància mínima de seguretat que ha d'haver-hi entre ambdós *Mobs*. També hi ha un tercer paràmetre (a banda del pes) amb el que es pot definir la predicció de la posició (en segons), el qual permetrà al *Mob* conèixer la posició de l'amenaça transcorregut aquest temps, i per tant calcular cap a on ha de fugir. Tot i semblar prou útil (paràmetre de predicció de temps en el SFPursuit) no ho és quan predador i presa estan prou aprop, ja que el vector predit varia més ràpidament quan més propers siguin i això fa que la presa faci un moviment en forma de ziga-zaga fugint del predador, en lloc de fugir en la direcció contrària, un comportament totalment allunyat de la realitat.

- **Estratègia de perseguir**: Aplicada als lleons i llops. En ambdós *Mobs* funciona de forma que, mitjançant l'script particular de cadascun, s'introdueix la presa en el paràmetre corresponent de *SteerForPursuit* i se li dóna un pes significatiu respecte els altres scripts. D'aquesta manera el predador persegueix la presa fins que l'assoleix i la mata.

La figura 37 detalla els components de les zebres: els scripts d'*Steering*, l'script propi i la FSM amb els quatre estats del *Mob* (*Thirsty*, *Hungry*, *Wander* i *Evasion*). La figura 38 detalla els components dels lleons, que són els mateixos que les zebres amb la diferència del (*SteerForPursuit*) per part de l'*Steering* i l'absència de l'estat *Evasion* a la FSM.

En segon lloc es presenten els diagrames de classes dels *Mobs* del bosc. La figura 39 detalla els components dels conills: els scripts d'*Steering*, l'script propi i la FSM.

En relació als dos BTs implementats (llop i pirata) s'han utilitzat els nodes de *PandaBT* especificats a la secció 5.1.4

Per a implementar la sincronització entre els BTs dels pirates s'han implementat una sèrie de booleans tant a l'script principal del pirata com a la memòria compartida que permeten activar i desactivar les sincronitzacions grupals.

En primer lloc, quan un pirata sobrepassa el rang de visió amb el jugador (subarbre *WatchingIdle*) s'activa el booleà corresponent a la *blackboard* que indicarà als pirates que es trobin en el rang de rodejar que han de demanar la seva posició a

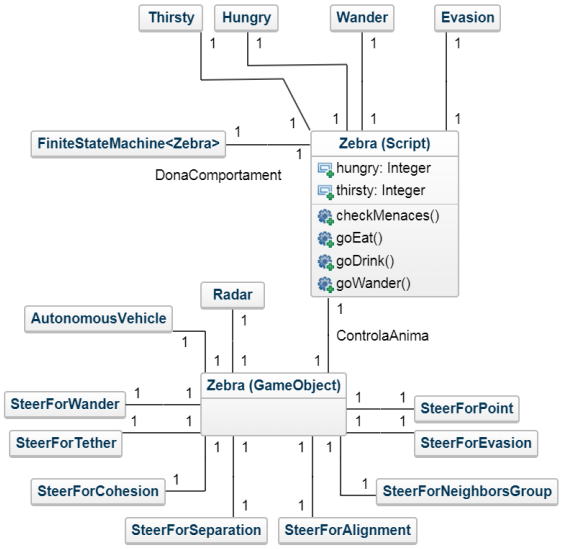


Figura 37: Diagrama de les classes del *Mob Zebra*

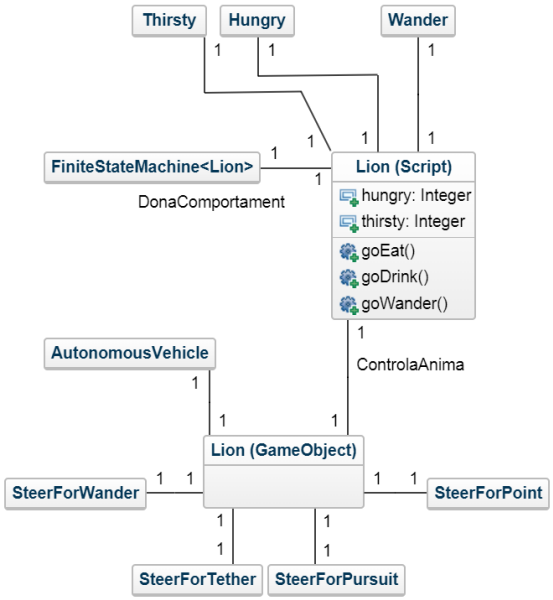


Figura 38: Diagrama de les classes del *Mob Lion*

la memòria compartida. Per altra banda també activa dos booleans interns que evitarà que executi el subarbre *SurroundHero*, ja que aquest pirata ja està rodejant el jugador.

Quan s'executi el BT dels altres pirates i validin si han d'entrar al subarbre *SurroundHero*, es comprovarà si el booleà de la *blackboard* està activat i si estan a la distància adequada (com s'ha dit, han d'estar dintre de la distàndia de rodejar el jugador, més gran que la de visió). Els que acompleixin ambdues condicions retornaran un *Success* que els permetrà demanar la posició a la *blackboard*.

Fins que no arribin a situar-se a la posició que se'ls hagi assignat no sortiran del

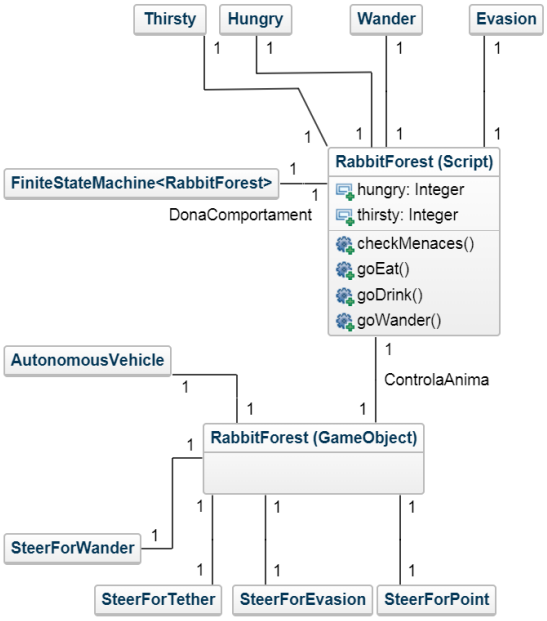


Figura 39: Diagrama de les classes del *Mob RabbitForest* (*RabbitForest* en lloc de *Rabbit* perquè la classe *Rabbit* ja l'utilitza el *Mob* de la granja).

```
tree("Root")
    fallback
        tree("SurroundHero")
        tree("Attack")
        tree("Follow")
        tree("WatchingIdle")
        tree("Patrol")

tree("SurroundHero")
    sequence
        checkSurroundHero
        surroundHero


tree("Attack")
    sequence
        checkAttack
        checkPirateLife
        attackPlayer


tree("Follow")
    sequence
        checkFollow
        alertFollowClosestPirates
        chasePlayer


tree("WatchingIdle")
    sequence
        checkWatchingIdle
        beWatching
        beIdle


tree("Patrol")
    sequence
        checkPatrol
        tree("Needs")
        moveToRandomPoint


tree("Needs")
    sequence
        satiateThirsty
        satiateHungry
```

(a) BT implementat al llop

```
[Task]
public void checkAttack()
{
    isAttack = isInAttackRange(LevelController.instance.Hero.gameObject);
    Task.current.Complete(isAttack);
}

[Task]
/**
 * Check player life and attack if possible
 */
public void attackPlayer()
{
    transform.LookAt(LevelController.instance.Hero.gameObject.transform.position);
    if (LevelController.instance.Hero.isDead())
    {
        Task.current.Fail();
    }
    else if (_nextAllowedAttackTime <= Time.time)
    {
        LevelController.instance.Hero.TakeDammage(_attackForce);
        _nextAllowedAttackTime = Time.time + _attackSpeed;
        activate("standBiteTrigger");
        activate("standBiteBool", true);
        Task.current.Succeed();
    }
    else
    {
        Task.current.Succeed();
    }
}
```

(b) Primera condició i acció del subarbre d'atac del llop

Figura 40: BT implementat al llop i exemple de nodes fulla.

subarbre *SurroundHero*. Quan hi arribin s'activaran els mateixos booleans que en el primer pirata, així es desactivarà la cerca del jugador, ja que ha estat trobat.

Quan el jugador s'apropi massa a un pirata sobrepassant el rang de perseguir-lo, aquest correrà cap al jugador per atacar-lo i avisarà a tots els que també estiguin

rodejant el jugador amb ell mitjançant un altre booleà de la *blackboard*. Un cop matin el jugador o aquest hagi escapat dels pirates, aleshores es desactiven els booleans de la *blackboard* i els interns per tal que tots els pirates tornin a fer el *Wander*.

La figura 41 detalla els components dels llops: l'script d'*Steering* i el BT. Com es pot observar aquest *Mob* és el primer en incorporar un BT, que li permet eliminar la FSM i reduir al mínim els scripts d'*Steering*.

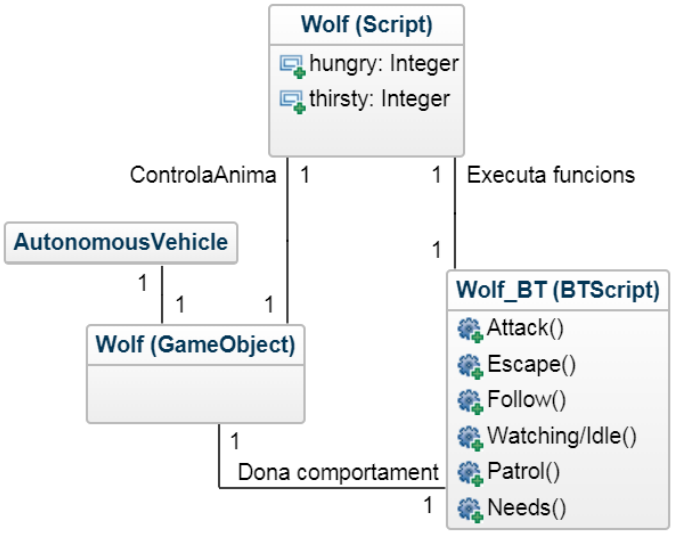


Figura 41: Diagrama de les classes del *Mob Wolf*.

Per últim, la figura 42 detalla els components dels pirates: els scripts d'*Steering* i el BT amb la memòria compartida.

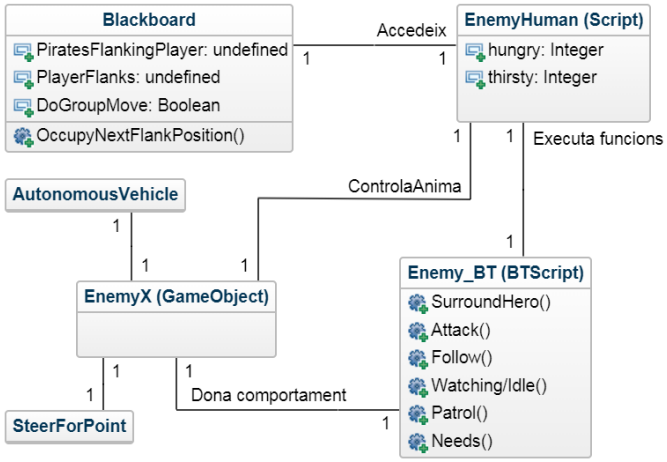


Figura 42: Diagrama de les classes del *Mob Pirate* (*EnemyHuman* és la classe del pirata, i la *X* pot ser 1 o 2, perquè s'han implementat dues figures 3D diferents).

5.4 Estratègies especials utilitzades en el desenvolupament del joc

Les diferents estratègies i tècniques dissenyades i implementades no han estat suficient en determinats casos, degut a que hi ha hagut certes situacions que han requerit d'una ajuda extra per tal de ser dutes a terme sense alguns problemes generats per les limitacions del software de *UnitySteer* o (com l'exemple d'anar variant la distància del *Tether*), o simplement han necessitat ser complementades per a desenvolupar la tasca correctament (lo del llac).

L'estratègia del *Wander* + *Tether* implementada amb *UnitySteer* té un problema que cal corregir mitjançant un script. El *SteerForTether* implementa un radi que defineix l'àrea màxima assolible pel *Mob*, fora d'aquesta àrea el *Mob* intentarà tornar-hi. El problema sorgeix quan s'aplica el *SteerForWander*, que genera un moviment constant i normalment circular. Aquest moviment circular acaba topant amb la distància màxima de *Tether*, aleshores el *Mob* vol tornar a entrar dins l'àrea. Quan està dins d'aquesta àrea ja no afecta, llavors torna a començar el moviment de *Wander* circular fins que torna a sobrepassar l'àrea del *Tether*. Aquest moviment d'anar i tornar entre ambdues estartègies es fa cada cop més ràpid fins que es genera un moviment en forma de ziga-zaga al llarg de tota l'àrea circular definida per l'*SteerForTether* (veure figura 43).

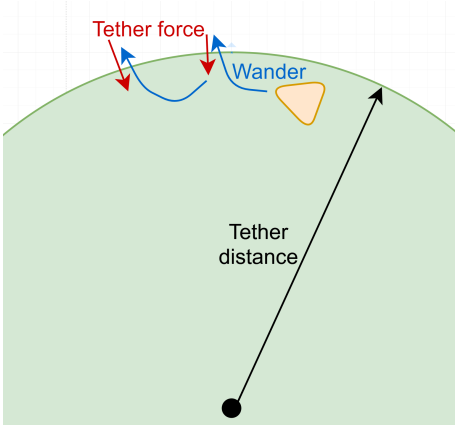
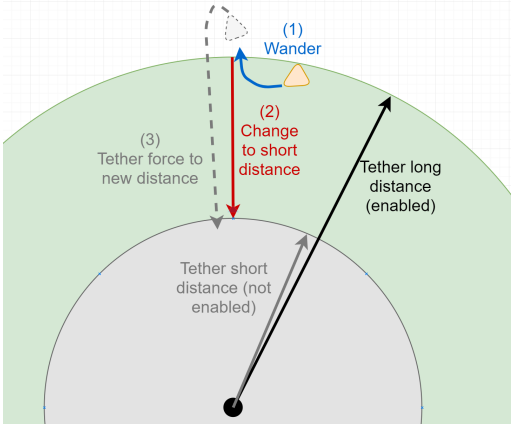


Figura 43: Problema generat per ambdues estratègies al juntar-se.

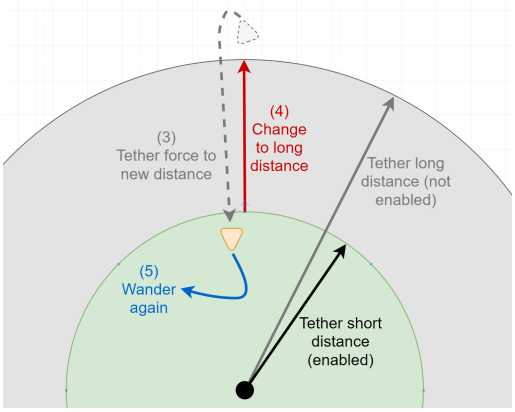
La solució a aquest problema consisteix en modificar la distància cada cop que l'assoleixi el *Mob*, com es pot veure a la figura 44.

Amb l'objectiu d'afegir realisme al *Wandering* dels *Mobs*, s'ha implementat un mètode que desactiva el moviment i només l'activa amb una probabilitat del 0.1%, ja que la major part del temps els animals estan quiets, descansant. També queda dissimulat el constant moviment circular del *SteerForWander*, molt poc realista.

Per simular l'efecte de beure ha calgut implementar diversos nodes (veure figura 45a que utilitzen els *Mobs* per a poder saciar la sed. El *Mob* escull el més proper i s'hi desplaça. El *Collider* de *Unity3D* fa que dos objectes que vulguin estar al mateix lloc topin i no deixin de moure's, així que per evitar que dos *Mobs* vagin al mateix node i topin, quan un *Mob* està molt aprop del node comprova si hi ha



(a) (1) Fa *Wander* i sobrepassa la distància màxima del *Tether*. (2) En sobrepassar el límit el canvia a la distància curta. (3) La força del *SteerForTether* fa que el *Mob* torni al límit establert, la distància curta en aquest cas.



(b) (4) Quan el *Mob* sobrepassa la nova distància es torna a habilitar la llarga. (5) El *Mob*, lluny de la distància límit, torna a fer *Wander* fins que es torni a trobar el límit de l'àrea gran, aleshores es tornarà a repetir el procés.

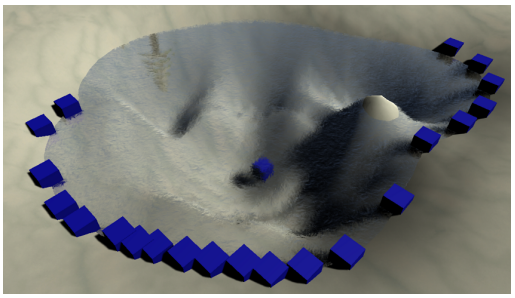
Figura 44: Solució al ziga-zaga de la figura 43. Amb aquesta solució s'evita el moviment de ziga-zaga perquè el límit mai és el mateix pel *Mob*, es van intercanviant a mesura que els assoleix.

algun altre *Mob* del mateix tipus molt aprop seu, gaire bé topant. En cas afirmatiu descarta aquell node i va al següent node més proper, fins que troba un lliure.

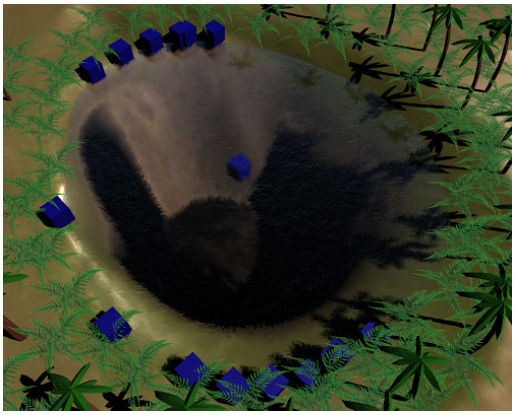
Aquesta estratègia ha estat especialment implementada pel llac de la figura 45b. S'han situat pocs node al llac. Així, a causa de la poca quantitat de nodes que hi ha, a la seva distribució i a la posició dels *Mobs* a l'escena de la platja, les probabilitats de que topin dos zebres o dos lleons és relativament alta. És majoritàriament degut a les zones en les que es troben les zebres i els lleons. Qualsevol punt de les seves àrees dóna com a node d'aigua més proper gaire bé sempre el mateix node, fet que provoca que si dues zebres (o dos lleons) van a beure aigua al mateix temps les probabilitats de que vagin al mateix node siguin altes.
A l'escena *Forest*, les probabilitats que succeeixi això són gaire bé nul · les ja que els *Mobs* tenen més espai (més dispersos, a distàncies diferents dels nodes), més nodes, i només són dels conills i els pirates, ja que els llops tenen el seu propi llac a l'altra banda del mapa (totes les zebres al mateix lloc, a una distància semblant dels nodes, igual amb els lleons).

També s'ha introduït un node al mig de cada llac, enfonsat, que els *Mobs* enfoquen quan estan bebent aigua. Amb això i l'animació de menjar s'aconsegueix que baixin el cap, fent l'efecte d'ajupir-se a beure aigua, que afegeix una petita dosi de realisme al joc.

Una altra estratègia especial afegida per agregar realisme ha estat el control de l'aliment dels herbívors (plantes rodejant el llac de la figura 45b). Quan un herbívor troba una planta per menjar aquesta inicia un compte enrere i desapareix (l'objecte és destruït) en uns pocs segons, simulant que ha estat menjada. Si el *Mob* continua



(a) Nodes d'aigua d'un llac de l'escena *Forest*



(b) Nodes d'aigua del llac de l'escena *Beach*

Figura 45: Nodes utilitzats pels *Mobs* a l'hora de beure aigua. Pintats en blau per a poder-los veure, normalment són transparents.

amb gana, busca més plantes.

El problema que sorgeix amb aquesta estratègia és que, al final, els *Mobs* herbívors es quedaran sense plantes. Per solucionar-ho s'ha aplicat a les plantes un script que fa el compte enrere esmentat abans però no destrueix la planta, només el fa desaparèixer al cap d'uns segons i el deixa inhabilitat. El següent herbívor que faci una cerca de plantes comprovarà només les que estiguin habilitades.

En desaparèixer l'herba o la planta, iniciarà un altre compte enrere més llarg que l'anterior. Quan s'acabi farà reaparèixer la planta i l'habilitarà de nou.

6 Resultats i Simulacions

En aquest capítol es mostren els resultats de les diferents simulacions realizades amb els diferents tipus d'animals i els humans.

El joc comença amb les tres interfícies de les figures 5, 6 i 7 de la secció 2.1.4.

En el moment que es pot començar a jugar es pot escollir entre les escenes de la figura 4. El jugador pot romandre a la ciutat principal (4a) per a parlar amb en *Lord Barus* o es pot desplaçar cap a la granja (4d), cap a la platja (4c) o finalment cap al bosc (4b).

6.1 Ocells

Quan el jugador entra a la platja es poden observar a tots els *Mobs* en la seva posició inicial.
Els ocells es troben tots junts (veure figura 46a). Es poden observar les connexions entre ells (línies rosades) que utilitza l'estratègia grupal d'*Steering* per a tenir en compte els veïns. Les circumferències de color blau fosc indiquen la distància a la que qualsevol *Mob* pot ser detectat, forma part de tots els que implementen qualsevol *script* d'*Steering*. Finalment, les línies cyan que es poden observar breument indiquen la distància que utilitza el *Radar* per a detectar els altres *Mobs*.

La figura 46b permet observar el desplaçament dels ocells quan han passat uns instants d'haver entrat a l'escena.
Es pot veure com influeix l'estratègia de *Wander* continuada en el moviment aleatori, ja que els ocells descriuen un moviment circular constant.
L'estratègia grupal propicia el moviment conjunt dels ocells, però es pot observar que no es mouen tots junts sinó que s'han dividit en dos grups d'ocells. És degut a que aquesta estratègia té en compte els *Mobs* més propers ja que dóna un pes més baix als que es troben lluny. Això provoca que, quan avancen pel moviment del *Wander*, els ocells acabin per dividir-se en grups per la llarga distància que hi ha amb els ocells més llunyans, que acaben per no ser tinguts en compte.

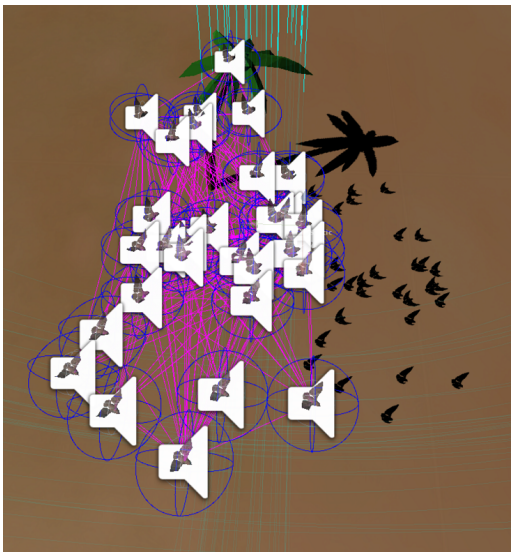
Transcorreguts uns segons, es pot observar que l'efecte anterior s'intensifica, provocant que els dos grups d'ocells inicials vagin perdent cada cop més membres, com es pot observar a les figures 47a i 47b.

Aquest comportament està fortament influït per la velocitat dels *Mobs*, que es tradueix en força pels càlculs dels vectors de moviment. Quanta més velocitat tinguin més força s'aplicarà als vectors de moviment, i per tant més fàcilment es pot perdre la connexió entre els ocells.

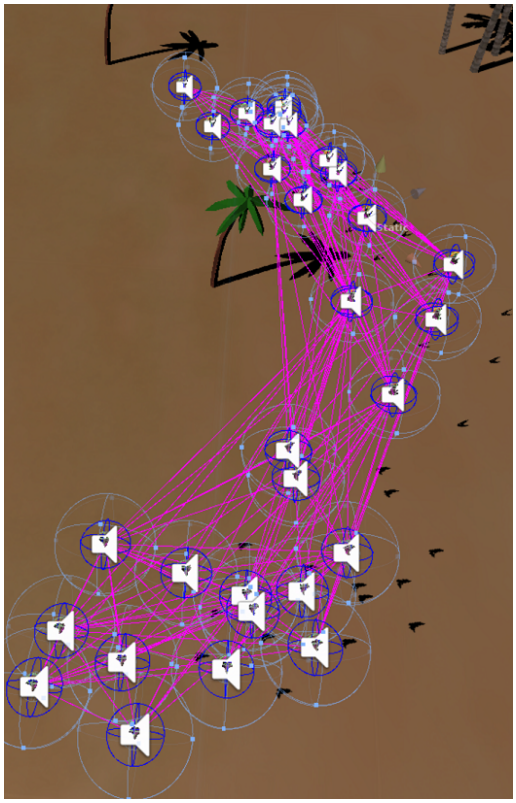
6.2 Zebres

Les zebres utilitzen diferents estratègies d'*Steering*.

En primer lloc, com es pot observar a la figura 48a inicialment es troben totes juntes, unides per l'estratègia grupal.

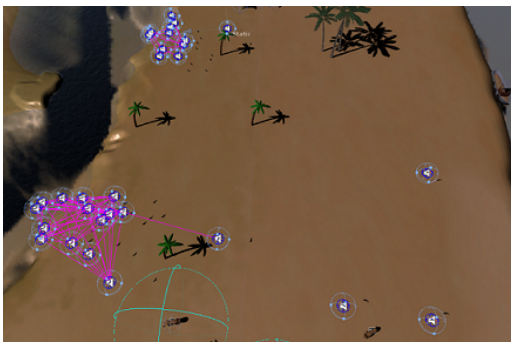


(a) Posició inicial dels *Mobs* al entrar a la platja.



(b) Moviment inicial dels *Mobs* transcorret un segon.

Figura 46: Posicions inicials dels ocells.



(a) Ocells en dos grups marcats i uns altres pocs separats dels grup.

(b) Encara dos grups però més dividits.

Figura 47: Posicions dels ocells després d'uns segons.

Transcorreguts uns segons algunes zebres se separen del grup (veure figura 48b, fent petits desplaçaments, però a diferència dels ocells una zebra no perd de vista les demés zebres. Aquest comportament és degut a l'algoritme implementat que obliga a les zebres a estar la major part del temps quietes i moure's només de tant en tant.

Això, junt amb l'estratègia de *Wander + Tether*, provoca que la probabilitat de que surtin de la distància del *Radar* siguin molt baixes.

(a) Posició inicial de les zebres en entrar a la platja.

(b) Moviment inicial de les zebres transcorreguts uns segons.

Figura 48: Posicions inicials de les zebres.

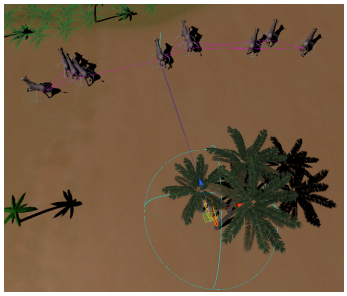


Figura 49: Problema generat per ambdues estratègies al juntar-se.

(a) Moviment de fugida al veure l'amenaça

(b) Zebra apropant-se al node d'aigua (cubs blaus) més proper.

(c) Zebra simulant que beu aigua.

Figura 50: Estratègies utilitzades per fugir, beure aigua i menjar.

Quan un lleó té gana persegueix a la zebra més propera. Aquestes fugen en la direcció contrària, tal i com es pot observar en la figura 49.

En aquest cas l'estratègia grupal perd força per cedir-la a la de fugir, tot i que se segueixen tenint en compte entre elles. Això propicia una fugida lleugerament grupal, observable només quan les zebres estan juntes i dintre de l'àrea permesa, on el *Tether* no té pes al càlcul del vector final i el vector de fugida és gaire bé paral · lel en totes les zebres, aleshores es pot observar una breu fugida grupal fins que sobrepassen l'àrea permesa.

L'algoritme implementat a la zebra per saciar la sed es basa en trobar el node d'aigua més proper i apropar-s'hi, com s'observa a la figura 50a.Un cop assolit, fixa el vector *Forward* al node enfosant al mig del llac i executa l'animació de menjar. D'aquesta forma es pot fer una simulació creïble del moment en què una zebra beu aigua (veure figura 50b).

6.3 Lleons

Les estratègies d'*Steering* utilitzades al lleons són molt semblants a les de les zebres, diferint en l'aspecte grupal (no utilitzen l'estratègia grupal) i a l'hora de menjar.

Inicialment els lleons es troben dispersos per la seva zona (veure figura 51a), fent el mateix *Wander + Tether* que les zebres, però individual.
També realitzen la mateixa acció a l'hora de beure aigua.

Quan han de saciar la gana aleshores utilitzen l'estratègia de perseguir, com es pot veure a la figura 49, on el lleó ha fixat l'objectiu (la zebra més propera) i la persegueix fins que està suficientment aprop com per atacar-la (fa l'animació d'atacar saltant i li treu vida (veure figura 51b). Quan l'ha atacada suficient aleshores la zebra mor (executa l'animació de morir) i el lleó ha satisfet la gana.
L'únic problema radica en què no es veu com el lleó es menja la zebra ja que l'animació de morir dura només dos segons (veure figura 51c).

(a) Posició inicial de les zebres al entrar a la platja.

(b) Moviment inicial de les zebres transcorreguts uns segons.

(c) Moviment inicial de les zebres transcorreguts uns segons.

Figura 51: Posicions inicials de les zebres.

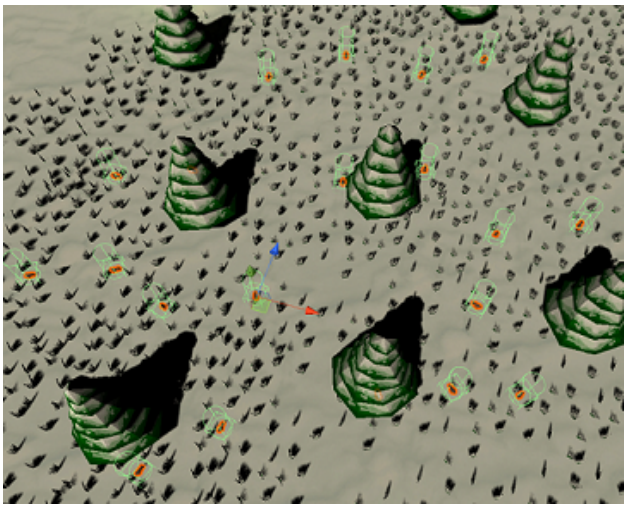
6.4 Conills

Els conills del bosc són relativament semblants a les zebres pel que fa al moviment i comportament, amb la variació de l'estratègia grupal, que no s'aplica a aquest *Mob*.

Inicialment els conills es troben a la zona de plantes (veure figura 52a).

Realitzen el *Wander + Tether* dintre de la zona de plantes, de la que només surten si han d'anar a beure aigua al llac, de la mateixa forma que ho fan les zebres, ja que ambdós tenen els mateixos algoritmes aplicats per dues implementacions idèntiques de la màquina d'estats. També fan el mateix quan han de saciar la gana, com es pot veure a la figura 52b.

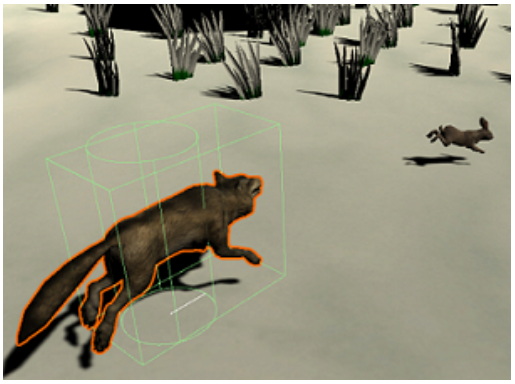
L'altra estratègia implementada és la de fugir. Tal i com es pot observar en la figura **??** el conill fuig del llop a una certa distància, de la mateixa forma que ho fan les zebres amb el lleó. Tot i així ambdós depredadors són més ràpids que les dues preses, així que finalment l'atrapa (veure figura 53b).



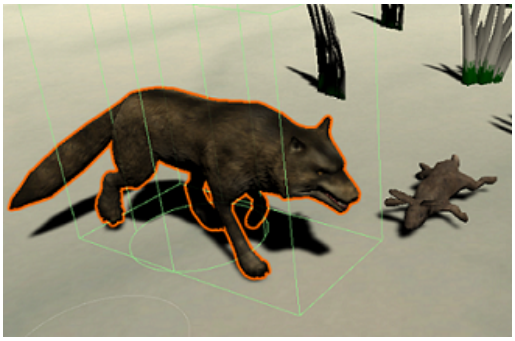
(a) Hàbitat dels conills al bosc.

(b) Acció del conill de menjar una planta.

Figura 52: Hàbitat dels conills (esquerra) i conill menjant (dreta).



(a) Conill fugint.



(b) Conill mort pel llop.

Figura 53: Conill fugint del llop (esquerra), morint finalment (dreta).

6.5 Llops

Els llops estan implementats amb BTs, independents entre ells.

Inicialment els llops es troben a l'altra banda de l'illa, com es pot veure a la figura 54.
No tenen implementada la mateixa estratègia de *Wander + Tether* que els *Mobs* vists fins ara, sinó que conserven la que tenien anteriorment, basada en un moviment continu entre diferents nodes de patrullatge. Cada cop que un llop arriba a un node escull el següent aleatòriament i s'hi desplaça (veure figura 54).

Per a saciar la gana fan el mateix que els lleons, busquen l'herbívor més proper i el persegueixen fins a matar-lo, com es veu a la figura 53.
Tal i com es pot veure a la figura 55a els llops localitzen el node d'aigua més proper i van a beure-hi aigua. També s'utilitza un node enfonsat que fa la mateixa funció que amb els *Mobs* de la platja.

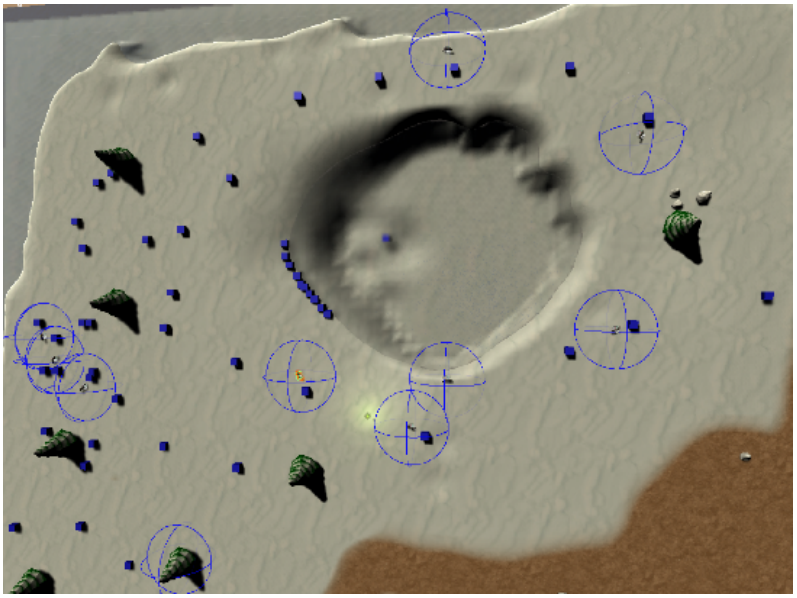
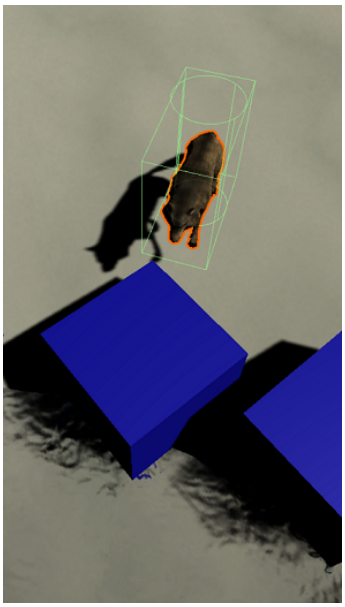
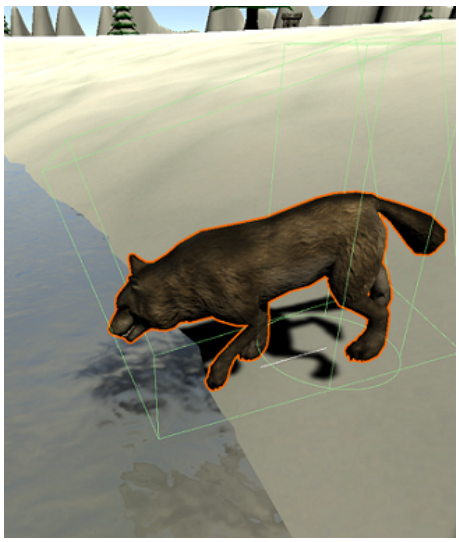


Figura 54: Territori dels llops. Els llops estan rodejats per les línies blaves que formen una esfera. Els cubs blaus són els nodes de patrullatge i els nodes d'aigua (els que estan a la vorera del llac i el que està enfonsat)

.



(a) Dirigint-se cap al node d'aigua.



(b) Llop bevent. No té cap animació com les zebres o els conills que ajupi el cap o es pugui simular que beu aigua, així que executa l'animació d'estar-se quiet normal.

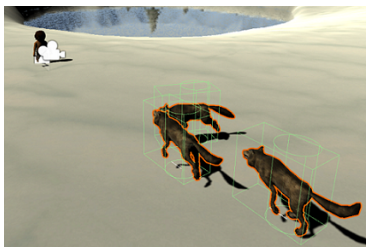
Figura 55: Llop utilitzant un node d'aigua per saciar la set, enfocant el seu vector *Forward* cap al node enfonsat del llac

La interacció amb el jugador és més complexa.
Com es pot veure a la figura 56a quan un llop (del tipus que sigui) es troba amb el

jugador, l'enfoca i s'està quiet, esperant a que el jugador reaccioni. Quan s'apropa suficientment al llop com per sobrepassar la distància aleshores escapa si és un beta, com es pot veure a la figura 56b. Ara cerca l'*alpha* més proper, però en el camí es troba dos llops *beta*, suficients per atacar al jugador (si encara es troba prou aprop d'ells), i s'hi llancen, com es pot observar a la figura 56c.

(a) Llop *beta* vigilant el jugador.

(b) Llop *beta* fugint.



(c) Grup de llops *beta* perseguint al jugador per atacar-lo.

Figura 56: Seqüència d'accions d'un llop *beta* desde que veu el jugador per primer cop, passant per la cerca d'altres llops, fins que els troba i ataquen al jugador.

6.6 Pirates

Els pirates, tot i haver estat implementats amb BTs com els llops i amb les mateixes funcions, només han estat considerats pel comportament generat pels BTs compartits amb la *blackboard*.

Inicialment els pirates es troben a l'altre part del bosc, aprop del pirata líder *John The Builder* (veure figura 57a).

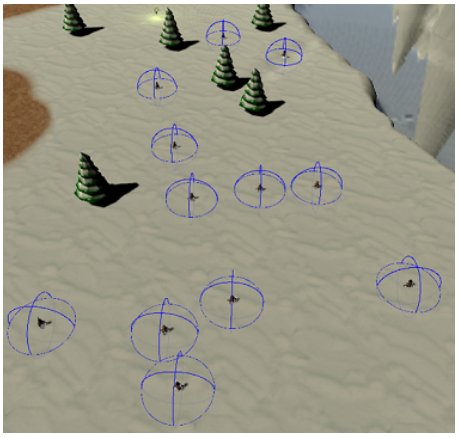
A les figures 57b, 58a i 58b es poden veure els diferents passos de formació de flancs i atac.
A la figura 57b el jugador encara no s'ha apropat prou com per a que algun pirata sobrepassi la circumferència marcada en color cyan.
Quan el jugador s'ha apropat una mica més es pot observar com el pirata que estaba més aprop s'ha quedat al lloc on es trobava prèviament, mentre que els demés pirates que es trobaven dintre del rang verd (distància de rodejar al jugador) van a buscar cadascun la seva posició dintre del flanc. Els pirates exteriors al rang verd no es tenen en compte.

Es pot observar els cercles grisos desocupats al costat del flanc lila.
Tot i que haurien d'estar ocupats, a la pràctica no es pot dur a terme tan rigorosament com a la teoria. L'algoritme s'executa correctament, però degut a l'estructura del BT, quan el pirata veu el jugador entra el subarbre de vigilar, invalidant la posició a la que hagués d'anar ja que vigilar al jugador passa a ser prioritari. És per això que, tot i estar molt aprop dels cercles grisos no estan just a sobre, perquè s'han apropat fins que han topat amb la distància de visió del jugador.

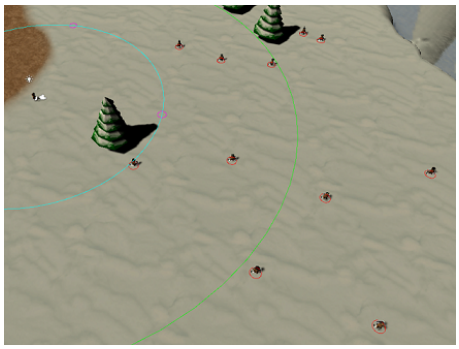
Després de fer diverses simulacions s'ha comprovat que aquest comportament simula millor la realitat, perquè quan un pirata veu el jugador s'atura en lloc d'anar

fins el seu lloc corresponent dins del flanc, ja que sinó el pirata es desplaça fins al seu lloc sense tenir en compte si està dins del rang de visió, fet que treu realisme a la implementació.

Quan el jugador sobrepassa la distància de perseguir d'un pirata aleshores aquest avisa a tots els demés mitjançant la *blackboard* per a què persegueixin al jugador (veure figura 58b).
Finalment, si el jugador escapa, ha de sobrepassar la distància de rodejar delsp irates (circumferència verda) per a què aquests deixin de perseguir-lo i tornint al moviment de *Wander*.

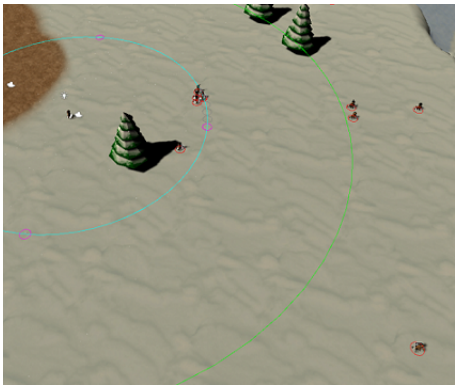


(a) Posició inicial dels pirates al bosc.



(b) Pirates (encerclats en vermell) i jugador (distàncies i flancs amb els mateixos colors que a la figura 27a).

Figura 57: Posicions inicials dels pirates i posicions abans de que cap pirata vegi el jugador.



(a) Pirates posant-se en formació. En gris les posicions dels flancs. El primer pirata es troba just al costat de l'ombra de l'arbre proper al jugador.

(b) Pirates perseguint el jugador.

Figura 58: Formacions i atac dels pirates.

6.7 Videos de les simulacions

Per a poder observar amb més detalls les simulacions s'han fet una sèrie de videos dels comportaments més importants dels *Mobs*.

Tots al canal de Youtube següent:

`https://www.youtube.com/channel/UCxb3tk3jEipZdmZm4bEMYMw`

7 Conclusions i feina futura

7.1 Conclusions

En aquest treball final de grau s'han assolit els objectius marcats en el seu començament. És a dir, s'han analitzat, seleccionat, dissenyat i desenvolupat diferents tècniques i estratègies d'intel · ligència artificial aplicades als personatges no controlats pel jugador del videojoc Fracsland. S'ha aconseguit generar moviments i comportaments més realistes, propis d'humans capaços d'actuar conjuntament o d'animals que interactuen entre ells per moure's en grup o individualment en un ecosistema.

El control dels diferents moviments i comportaments individuals dels *Mobs* s'han dut a terme mitjançant l'ús de la ***Navigation Mesh***, útil per a definir i controlar les zones per les que es poden moure els *Mobs*; una combinació de d'***Steering*** i **màquines d'estats**, que ha permès modelar tot tipus de moviments i comportaments individuals com beure, menjar, fugir, perseguir i deambular, englobant-los dintre d'estats, fet que en facilita la manipulació i tractament. En contrapartida a l*Steering* i les màquines d'estats s'han aplicat els **arbres de comportament** per a poder gestionar també els comportaments individuals però d'una altra forma més precisa i complexa.

El control dels comportaments grupals ha estat modelat, per una banda, amb l'estratègia grupal d'*Steering*, que permet implementar un comportament grupal realista, propi de manades d'animals, i amb l'estratègia de la **sincronització d'arbres de comportament** per altra banda, que permet configurar d'una forma més precisa i la sincronització de grups per a comportaments grupals més complexos, com humans que es coordinen entre ells.

Després d'haver provat les diferents tècniques de modelatge del moviment s'ha arribat a la conclusió que l'*Steering* és una tècnica molt útil a l'hora de modelar el moviment ja que no executa un moviment o un altre, sinó que calcula el pes de totes les estratègies de moviment actives i n'extreu un vector de direcció final. Aquesta estratègia de càlcul del moviment ofereix més realisme ja que no genera moviments ben marcats sinó moviments amb translacions més o menys suaus entre ells degudes als diferents pesos.

Comparant les dues tècniques de modelatge del comportament, s'ha arribat a la conclusió que les màquines d'estats són molt útils a l'hora de modelar comportaments globals, ja que compten amb transicions entre estats, les quals permeten definir les condicions per a passar d'un estat a un altre, fent així possible un control general de tot el comportament. Per altra banda, no són útils a l'hora d'implementar comportaments complexes i massa específics que requereixin flexibilitat a causa de que les transicions restringeixen el pas entre comportaments, obligant a un estat concret a anar cap a un altre concret sempre.

En contrapartida a les màquines d'estat, els arbres de comportament sí que resulten molt útils a l'hora de modelar comportaments complexes i específics que requereixin flexibilitat perquè un sol arbre permet modelar diferents comportaments

sense transicions entre ells, ja que a diferència de les màquines d'estat, s'avalua l'arbre complet fins a trobar el comportament adequat donades unes condicions, aleshores executa aquest comportament. A més a més, els nodes intermitjos que té permeten crear comportaments molt complexes subdividits en una o més branques, basats en l'execució d'accions donades unes condicions prèvies. Per altra banda, aquesta contínua avaluació de l'arbre sencer i la necessitat de nodes intermitjos per tal d'executar accions fa que no sigui el millor mètode per a modelar comportaments globals, ja que no té una implementació senzilla ni tampoc transicions entre estats.

Si l'objectiu és modelar un comportament senzill i que englobi tots els estats globals que poden haver-hi aleshores el més adient és utilitzar una màquina d'estats. Si per altra banda l'objectiu és modelar un comportament complex i específic com per exemple una seqüència d'una baralla en la que en qualsevol moment s'ha de realitzar qualsevol acció (atacar, esquivar, fugir, etc.) aleshores cal utilitzar un arbre de comportament.

7.2 Línies futures

Tot i que tots els objectius relatius al motor IA del joc definits inicialment en el projecte s'han dut a terme, encara queden moltes parts, tant d'IA com d'altres àmbits relatius als videojocs que poden ser implementades per tal de millorar el joc de Fracsland.

7.2.1 Intel · ligència artificial

En primer lloc, les línies de continuació futures més inmediates serien les que fan referència a la IA: tècniques d'*Steering* i arbres de comportament.

Degut a que l'*Steering* és una tècnica que té en compte tots els vectors de totes les tècniques, cal ajustar molt bé els paràmetres fins a trobar la combinació correcte que permet dur a terme els moviments i comportaments desitjats. Una possible línia futura seria la modificació de tots aquests paràmetres per a optimitzar les tècniques d'*Steering* ja implementades.

L'altra possible línia futura seria dotar de comportament més complexe i optimitzar els arbres de comportament. En el cas dels llops es podria substituir per una màquina d'estats que controli els estats globals del llop i implementar diversos arbres de comportament, un per cada situació complexa, com la d'anar a cercar altres llops quan un troba al jugador. Així tindrien un ús més propi d'un arbre de comportament, en lloc de controlar amb un sol arbre tots els comportaments dels llops.

En el cas dels pirates es podrien dissenyar altres tècniques com tipus de formacions embarbussaments, simular interaccions humanes entre ells al trobar-se, caçar conjuntament llops i/o conills, etc.

7.2.2 Altres línies futures

A banda de tota la IA que pot ser millorada, també hi ha apectes que podrien ser optimitzats o altres elements que podrien ser afegits, com per exemple més *Mobs*, per fer els terrenys de la platja i el bosc més complets, fins i tot afegir algun altre escenari per a poder implementar noves tècnqiues de IA amb nous *Mobs*.

En última instància, una altra possible línia futura seria implementar el multijugador al Fracsland, per fer que tots els alumnes puguin jugar junts, cooperativament. Per aiò, caldria modificar el servidor per a permetre el multijugador i, a més a més, crear una instància del joc (a la màquina on corri el servidor) que sigui l'encarregada de comunicar-se amb totes les instàncies del joc creades en els clients a temps real.

Apèndix A: Manual tècnic

A.1 Instal · lació: Requeriments mínims i passos a seguir

Per a poder seguir desenvolupant aquest projecte en mode local cal disposar de:

- La versió de *Unity3D*: `Unity 2017.3.0p3 Personal (64bit)`.
- Un editor de text com per exemple: `Microsoft Visual Studio 2015`.
- El projecte de *Unity3D* de *Fracsland*.
- El servidor *stand alone* que contesta les peticions *JSON* del joc, guardat com un projecte de NetBeans. L'arxiu principal és la classe de Java `FracslandServer.java`.

Havent instal · lat *Unity3D* i descarregat el projecte i el servidor, cal modificar la ruta de l'arxiu `Constants.cs` del projecte i posar la direcció del servidor, és a dir, la direcció de la carpeta on es troba l'arxiu `FracslandServer.java` junt amb `/api/v1/` al final, que és on es crea el context del servidor.

Dins la classe Java cal posar també la direcció de la carpeta d'aquest arxiu (`FracslandServer.java`) i crear els contextos necessaris, tots amb la seva ruta, de forma que quedin: `pathToFolder+/api/v1/+context` i coincideixin amb les rutes de contextos de l'arxiu del joc `Constants.cs`. D'aquesta manera, quan s'executi l'arxiu Java, es crearà allà el context que utilitzarà el servidor per a respondre al joc.

Fet això només cal engegar el servidor i després executar el projecte de *Unity3D*.

Funciona de la mateixa forma si només es vol jugar al *Fracsland* en lloc d'executar el projecte. Cal configurar igual el servidor i el projecte, i finalment fer el *Build & Run* per obtenir l'executable del joc.

Referències

[1] Django full stack framework for Python, Last access June 2018.

[2] Begue, E.; PandaBT, Behavior Tree Scripting for Unity, http://www.pandabehaviour.com/, 2016, Last access June 2018.

[3] Navigation System in Unity3D, https://docs.unity3d.com/Manual/nav-NavigationSystem.html, Last access June 2018.

[4] Millington, I.; Funge J.; ARTIFICIAL INTELLIGENCE FOR GAMES, second edition, 2009, http://lecturer.ukdw.ac.id/mahas/dossier/gameng_AFG.pdf/, Last access June 2018

[5] J. Méndez, R.; UnitySteer 3.1, https://github.com/ricardojmendez/UnitySteer, 2016, Last access June 2018.

[6] 2D and 3D graphic engine, Latest version 2018.1.6, https://unity3d.com/, Last access June 2018.

[7] Muriel, C.; Fracsland: joc seriós per aprendre fraccions, http://diposit.ub.edu/dspace/bitstream/2445/104290/2/memoria.pdf, 2016.

[8] Champandard, A.J.; Dunstan P.; GameAIPro Capítol 6 The Behavior Tree Starter Kit.

[9] Pillosu, R.; Coordinating Agents With Behavior Trees, 2009, http://aigamedev.com/premium/presentations/coordination-behavior-trees/, Last access June 2018.

[10] Dawe, M.; Gargolinski, S.; Dicken, L.; Humphreys, T.; Mark, D.; GameAIPro Capítol 4 Behavior Selection Algorithms.

[11] Palacios, J.; Unity 5.x Game AI Programming Cookbook, Published by Packt Publishing Ltd., .2016

[12] Batut, C.; Belabas, K.; Bernardi, D.; Cohen, H.; Olivier, M.: User's guide to *PARI-GP*, pari.math.u-bordeaux.fr/pub/pari/manuals/2.3.3/users.pdf, 2000.

[13] Chen, J. R.; Wang, T. Z.: On the Goldbach problem, *Acta Math. Sinica*, 32(5):702-718, 1989.

[14] Deshouillers, J. M.: Sur la constante de Šnirel′man, *Séminaire Delange-Pisot-Poitou, 17e année: (1975/76), Théorie des nombres: Fac. 2, Exp. No.* G16, pàg. 6, Secrétariat Math., Paris, 1977.

[15] Deshouillers, J. M.; Effinger, G.; te Riele, H.; Zinoviev, D.: A complete Vinogradov 3-primes theorem under the Riemann hypothesis, *Electron. Res. Announc. Amer. Math. Soc.*, 3:99-104, 1997.

[16] Dickson, L. E.: *History of the theory of numbers. Vol. I: Divisibility and primality*, Chelsea Publishing Co., New York, 1966.

[17] Hardy, G. H.; Littlewood, J. E.: Some problems of 'Partitio numerorum'; III: On the expression of a number as a sum of primes, *Acta Math.*, 44(1):1-70, 1923.

[18] Hardy, G. H.; Ramanujan, S.: Asymptotic formulae in combinatory analysis, *Proc. Lond. Math. Soc.*, 17:75-115, 1918.

[19] Hardy, G. H.; Wright, E. M.: *An introduction to the theory of numbers*, 5a edició, Oxford University Press, 1979.

[20] Helfgott, H. A.: Minor arcs for Goldbach's problem, `arXiv:1205.5252v4 [math.NT]`, desembre de 2013.

[21] Helfgott, H. A.: Major arcs for Goldbach's problem, `arXiv:1305.2897v4 [math.NT]`, abril de 2014.

[22] Helfgott, H. A.: The ternary Goldbach conjecture is true, `arXiv:1312.7748v2 [math.NT]`, gener de 2014.

[23] Helfgott, H. A.; Platt, D.: Numerical verification of the ternary Goldbach conjecture up to $8.875 \cdot 10^{30}$, `arXiv:1305.3062v2 [math.NT]`, abril de 2014.

[24] Klimov, N. I.; Pil′tjaĭ, G. Z.; Šeptickaja, T. A.: An estimate of the absolute constant in the Goldbach-Šnirel′man problem, *Studies in number theory, No. 4*, pàg. 35-51, Izdat. Saratov. Univ., Saratov, 1972.

[25] Liu, M. C.; Wang, T.: On the Vinogradov bound in the three primes Goldbach conjecture, *Acta Arith.*, 105(2):133-175, 2002.

[26] Oliveira e Silva, T.; Herzog, S.; Pardi, S.: Empirical verification of the even Goldbach conjecture and computation of prime gaps up to $4 \cdot 10^{18}$, *Math. Comp.*, 83:2033-2060, 2014.

[27] Ramaré, O.: On Šnirel′man's constant, *Ann. Scuola Norm. Sup. Pisa Cl. Sci.*, 22(4):645-706, 1995.

[28] Riesel, H.; Vaughan, R. C.: On sums of primes, *Ark. Mat.*, 21(1):46-74, 1983.

[29] Rosser, J. B.; Schoenfeld, L.: Approximate formulas for some functions of prime numbers, *Illinois J. Math.*, 6:64-94, 1962.

[30] Schnirelmann, L.: Über additive Eigenschaften von Zahlen, *Math. Ann.*, 107(1):649-690, 1933.

[31] Tao, T.: Every odd number greater than 1 is the sum of at most five primes, *Math. Comp.*, 83:997-1038, 2014.

[32] Travesa, A.: *Aritmètica*, Col · lecció UB, No. 25, Barcelona, 1998.

[33] Vaughan, R. C.: On the estimation of Schnirelman's constant, *J. Reine Angew. Math.*, 290:93-108, 1977.

[34] Vaughan, R. C.: *The Hardy-Littlewood method*, Cambridge Tracts in Mathematics, No. 125, 2a edició, Cambridge University Press, 1997.

[35] Vinogradov, I. M.: Sur le théorème de Waring, *C. R. Acad. Sci. URSS*, 393-400, 1928.

[36] Vinogradov, I. M.: Representation of an odd number as a sum of three primes, *Dokl. Akad. Nauk. SSSR*, 15:291-294, 1937.